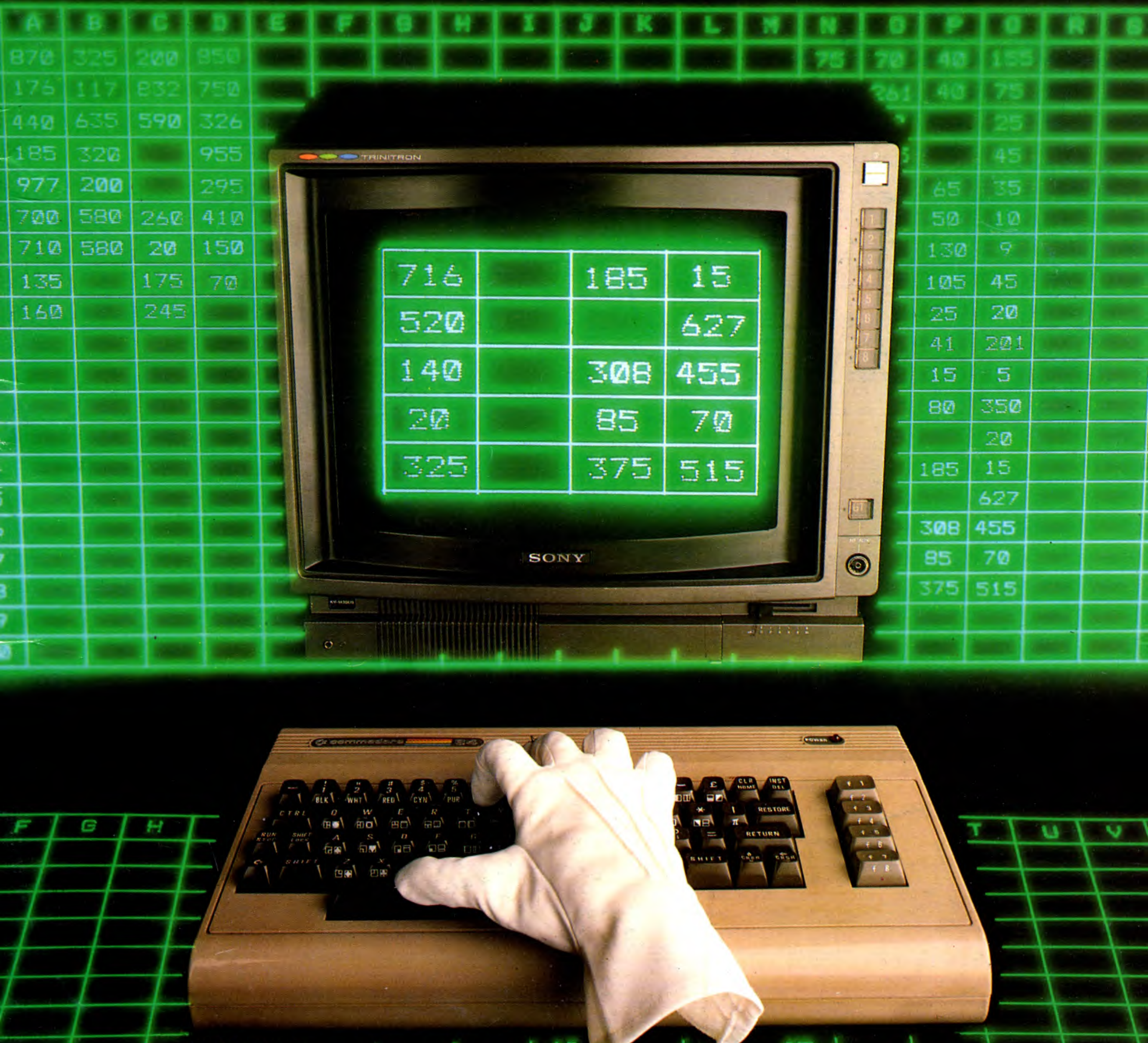
A MARSHALL CAVENDISH **38** COMPUTER COURSE IN WEEKLY PARTS

INPUT

LEARN PROGRAMMING - FOR FUN AND THE FUTURE



UK £1.00 Republic of Ireland £1.25 Malta [illegible]5c Australia $2.25 New Zealand $2.95

INPUT

Vol. 3 **No 38**

INDEX

The last part of INPUT, Part 52, will contain a complete, cross-referenced index. For easy access to your growing collection, a cumulative index to the contents of each issue is contained on the inside back cover.

PICTURE CREDITS

Front cover, Dave King. Pages 1177, 1178, 1180, 1183, Mohsen John Modaberi. Pages 1181, 1188, 1189, 1196, Peter Reilly. Pages 1185, 1190, Dave King. Pages 1192, 1194, 1196, Projection/Berry Fallon Design. Pages 1198, 1200, 1203, Peter Richardson. Page 1202, Berry Fallon Design. Pages 1204, 1206, 1208, Dave King.

Published by Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA, England. Typeset by MS Filmsetting Limited, Frome, Somerset. Printed by Cooper Clegg Web Offset Ltd, Gloucester and Howard Hunt Litho, London.



There are four binders each holding 13 issues.

HOW TO ORDER YOUR BINDERS

UK and Republic of Ireland: Send £4.95 (inc p & p) (IR£5.95) for each binder to the address below:
Marshall Cavendish Services Ltd, Department 980, Newtown Road, Hove, Sussex BN3 7DN

Australia: See inserts for details, or write to INPUT, Times Consultants, PO Box 213, Alexandria, NSW 2015

New Zealand: See inserts for details, or write to INPUT, Gordon and Gotch (NZ) Ltd, PO Box 1595, Wellington

Malta: Binders are available from local newsagents.

BACK NUMBERS

Back numbers are supplied at the regular cover price (subject to availability).

UK and Republic of Ireland:
INPUT, Dept AN, Marshall Cavendish Services, Newtown Road, Hove BN3 7DN

Australia, New Zealand and Malta:
Back numbers are available through your local newsagent.

COPIES BY POST

Our Subscription Department can supply copies to any UK address regularly at £1.00 each. For example the cost of 26 issues is £26.00; for any other quantity simply multiply the number of issues required by £1.00. Send your order, with payment to:

Subscription Department, Marshall Cavendish Services Ltd, Newtown Road, Hove, Sussex BN3 7DN

Please state the title of the publication and the part from which you wish to start.

HOW TO PAY: Readers in UK and Republic of Ireland: All cheques or postal orders for binders, back numbers and copies by post should be made payable to:
Marshall Cavendish Partworks Ltd.

QUERIES: When writing in, please give the make and model of your computer, as well as the Part No., page and line where the program is rejected or where it does not work. We can only answer specific queries – and please do not telephone. Send your queries to INPUT Queries, Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA.

INPUT IS SPECIALLY DESIGNED FOR:

The SINCLAIR ZX SPECTRUM (16K, 48K, 128 and +), COMMODORE 64 and 128, ACORN ELECTRON, BBC B and B+, and the DRAGON 32 and 64.

In addition, many of the programs and explanations are also suitable for the SINCLAIR ZX81, COMMODORE VIC 20, and TANDY COLOUR COMPUTER in 32K with extended BASIC. Programs and text which are specifically for particular machines are indicated by the following symbols:

SPECTRUM 16K, 48K, 128, and +
COMMODORE 64 and 128
ACORN ELECTRON, BBC B and B+
DRAGON 32 and 64
ZX81
VIC 20
TANDY TRS80 COLOUR COMPUTER

WILL YOU WALK INTO MY PARLOUR?

Release the starving Martian spider onto frantic Freddy when you follow through how an arcade-type game is written. In the next part you'll have the complete game

Freddy and the Spider from Mars (or the Window Cleaner's Nightmare) is a complete arcade-type game, building up over the next two parts of Games Programming.

In this first part, the graphics are set up and the program is initialized. In part two the working game will be produced by knitting the routines together.

THE GAME

The starting point for designing any game is an idea—some kind of plot or storyline, or maybe a character around which to build the game.

In this game Freddy is a window cleaner with an intense dread of spiders. He's been to his doctor and to innumerable specialists who cannot cure his condition. Things are so bad he now has a recurring nightmare, concerning a Martian spider (a particularly huge, hungry, nasty-looking variety of Freddy's favourite phobia), a collection of balloons, and his favourite hobby, archery.

Frequently he wakes up bathed in sweat after dreaming of being stranded on his ladder equipped, not with his usual bucket of dirty water and a wash leather, but with a supply of arrows. He has been trying desperately to burst balloons which, if allowed to reach the spider's cage above his head, unlock the doors which imprison the beast.

Help Freddy burst the balloons, or he'll come to a gruesome end as the spider's breakfast!

So much for the plot. As for the scoring, points are awarded for each balloon Freddy bursts. But bursting the lot doesn't bring relief—Freddy's torture continues at a higher level, with faster moving balloons. If he lets through three balloons, the Martian spider is released.

WRITING THE GAME

There are four objects that move about on the screen—the spider, which can move both horizontally and vertically; Freddy, who moves only vertically; the balloon currently on the screen, which also moves only vertically; and finally the arrow. This normally moves horizontally, but when Freddy ascends or descends the ladder and is holding the arrow it will have to move vertically with him.



The main objects of interest, though, are the spider and the balloon. These have many variables associated with them. A technique that proves useful here is to store all the variables in a one-dimensional array, and use a constant to reference a particular component.

The next stage is to think of how you are going to display the shapes on the screen. This involves considering all the different pictures that will be displayed during the course of the game. All the best computer games feature large, colourful shapes, so this is a very important stage in developing the program. In order to provide enough detail, Freddy will be constructed out of a three by two arrangement of UDG characters. Two by two pictures will be used to display the spider (there will be two of these, to make the animation more impressive), the balloon shape and a picture of a popped balloon. If you include two characters to display the arrow, and two more for the ladder, you end up having allocated 26 characters.

Part one sees the graphics set up and the game initialized.

GRAPHICS INITIALIZATION

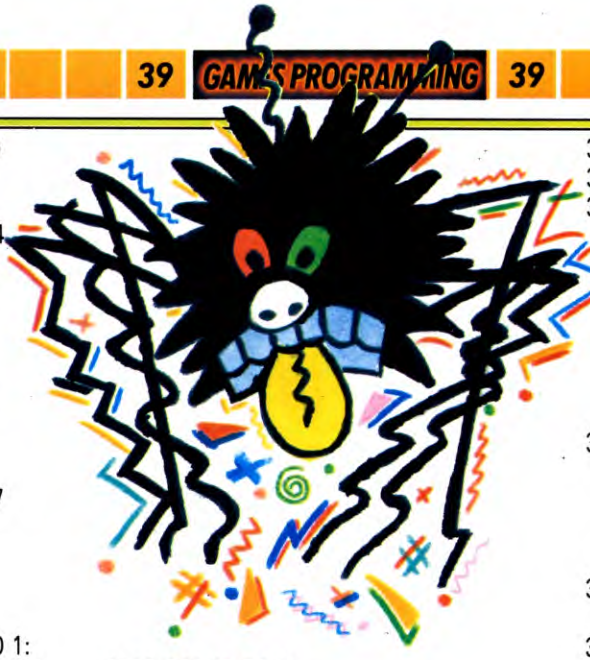
```
1000 DIM b(6): DIM s(7)
1010 LET xpos = 1: LET ypos = 2: LET
    colour = 3: LET points = 4: LET count = 5:
    LET maxcount = 6
1020 LET xinc = 3: LET yinc = 4: LET
    picture = 7
1030 LET dest = 65288
1040 FOR i = 0 TO 26*8 - 1: READ j: POKE
    dest + i,j: NEXT i
1050 DATA 15,63,127,255,255,255,255,127
1060 DATA 240,252,254,255,255,255,255,254
1070 DATA 127,63,63,31,15,7,3,6
1080 DATA 254,252,252,248,240,224,192,96
1090 DATA 32,96,255,255,96,32,0,0
1100 DATA 5,10,252,252,10,5,0,0
1110 DATA 1,64,17,40,16,0,0,161
```



```
1120 DATA 128,2,136,2Ø,8,Ø,Ø,133
113Ø DATA 161,Ø,Ø,16,4Ø,17,64,1
114Ø DATA 133,Ø,Ø,8,2Ø,136,2,128
115Ø DATA 48,48,48,48,111,111,48,48
116Ø DATA 12,12,12,12,246,246,12,12
117Ø DATA 7,31,49,57,127,112,237,255
118Ø DATA 224,248,14Ø,2Ø4,254,14,183,
   255
119Ø DATA 127,59,51,99,115,35,6,12
12ØØ DATA 254,1Ø8,1Ø2,51,49,25,24,48
121Ø DATA 7,31,49,51,127,112,235,255
122Ø DATA 224,248,14Ø,156,254,14,
   215,255
123Ø DATA 127,51,5Ø,27,25,5Ø,112,224
124Ø DATA 254,2Ø4,1Ø8,1Ø2,54,22,3,6
125Ø DATA 15,31,19,55,55,63,63,15
126Ø DATA 24Ø,248,248,252,252,252,
   252,24Ø
127Ø DATA 251,219,139,219,219,251,
   247,239
128Ø DATA 252,254,254,254,254,254,
   254,252
129Ø DATA 95,127,31,31,31,63,127,127
13ØØ DATA 188,188,188,188,188,124,
   252,248
131Ø LET hiscore = Ø
132Ø RETURN
```

```
1ØØØ DIM B(6),S(7),NU$(9)
1Ø1Ø XP = 1:YP = 2:PO = 4:CT = 5:MC = 6
1Ø2Ø XI = 3:YI = 4:PI = 7
1Ø22 DIM AD(14),EF(4),E(4),GJ(7),KL(4),
   MP(7),QT(7),UZ(14),SP(7),S1(4),S2(7),
   BL(4)
1Ø23 PMODE4,1:PCLS1:SG = PEEK(188)*256
1Ø24 GOSUB 15ØØ
1Ø25 GET(Ø,Ø) – (15,23),AD,G:PCLS1
1Ø26 GOSUB 152Ø
1Ø27 GET(Ø,Ø) – (15,7),EF,G:
   GET(Ø,Ø) – (7,7),E,G:PCLS1
1Ø28 GOSUB 15ØØ
1Ø29 GET(Ø,Ø) – (15,15),GJ,G:PCLS1
1Ø3Ø GOSUB 152Ø
1Ø31 GET(Ø,Ø) – (15,7),KL,G:PCLS1
1Ø32 GOSUB 15ØØ
1Ø33 GET(Ø,Ø) – (15,15),MP,G:PCLS1
1Ø34 GOSUB 15ØØ
1Ø35 GET(Ø,Ø) – (15,15),QT,G:PCLS1
1Ø36 GOSUB 15ØØ:SG = SG + 512:
   GOSUB 152Ø:SG = SG – 512
1Ø37 GET(Ø,Ø) – (15,23),UZ,G:PCLS1
1Ø38 GET(Ø,Ø) – (15,15),SP,G
1Ø39 GET(Ø,Ø) – (7,7),S1,G
1Ø4Ø GET(Ø,Ø) – (15,7),S2,G
1Ø41 PCLSØ:GET(Ø,Ø) – (7,7),BL,G
1Ø5Ø DATA 15,63,127,255,255,255,255,
   127
1Ø6Ø DATA 24Ø,252,254,255,255,255,
   255,254
1Ø7Ø DATA 127,63,63,31,15,7,3,6
1Ø8Ø DATA 254,252,252,248,24Ø,224,
   192,96
1Ø9Ø DATA 32,96,255,255,96,32,Ø,Ø
11ØØ DATA 5,1Ø,252,252,1Ø,5,Ø,Ø
111Ø DATA 1,64,17,4Ø,16,Ø,Ø,161
112Ø DATA 128,2,136,2Ø,8,Ø,Ø,133
113Ø DATA 161,Ø,Ø,16,4Ø,17,64,1
114Ø DATA 133,Ø,Ø,8,2Ø,136,2,128
115Ø DATA 48,48,48,48,111,111,48,48
116Ø DATA 12,12,12,12,246,246,12,12
117Ø DATA 7,31,49,57,127,112,237,
   255
118Ø DATA 224,248,14Ø,2Ø4,254,14,
   183,255
119Ø DATA 127,59,51,99,115,35,6,12
12ØØ DATA 254,1Ø8,1Ø2,51,49,25,24,48
```





```
1210 DATA 7,31,49,51,127,112,235,255
1220 DATA 224,248,140,156,254,14,
   215,255
1230 DATA 127,51,50,27,25,50,112,224
1240 DATA 254,204,108,102,54,22,3,6
1250 DATA 15,31,19,55,55,63,63,15
1260 DATA 240,248,248,252,252,252,
   252,240
1270 DATA 251,219,139,219,219,251,
   247,239
1280 DATA 252,254,254,254,254,254,
   254,252
1290 DATA 95,127,31,31,31,63,127,127
1300 DATA 188,188,188,188,188,124,
   252,248
1310 HS=0
1320 RETURN
1500 FOR CL=0 TO 1:FOR CH=0 TO 1:
   FOR L=0 TO 7:READ J:POKE SG+CL*
   256+CH+L*32,255-J:NEXT L,CH,CL
1510 RETURN
1520 FOR CH=0 TO 1:FOR L=0 TO 7:
   READ J:POKE SG+CH+L*32,255-J:
   NEXT L,CH
1530 RETURN
1600 DATA R6D8L6U8BR8,BR6ND8BR2,R6D4
   L6D4R6BR2BU8,R6D4NL3D4NL6BR2BU8,
   D4R6D4U8BR2,NR6D4R6D4L6BE8
1610 DATA D8R6U4L6U4BR8,R6ND8BR2,
   R6D8L6U8D4R6U4BR2,D4R6D4U8L6BR8
1620 FOR I=0 TO 9:READ NU$(I):NEXT
1625 DRAW "C1;S2"
1630 RETURN
1650 N$=STR$(NU):FOR I2=2 TO LEN(N$)
1660 DI=ASC(MID$(N$,I2,1))-48:DRAW
   NU$(DI)+"BR2":NEXT I2:RETURN
1700 COLOR0:LINE(178,2)-(200,7),PSET,
   BF:NU=HS:DRAW "C1;BM178,2":
   GOSUB 1650:RETURN
```

This section of program stores the DATA for the UDGs for the balloon and the spider in arrays B (or b) and S (or s)—the arrays are DIMensioned in Line 1000.

Lines 1010 and 1020 define the initial values of the pointers to the arrays, before the UDGs are set up. Finally, Line 1310 sets the high score to zero. This line has been placed here as it has to be executed only once during the program.

The Dragon/Tandy program has an extra routine starting at Line 1600 which DRAWs numbers on the high resolution screen.

GAME INITIALIZATION

```
3000 LET score=0: LET level=1
3010 LET my=15
3020 LET bl=15+5*level: LET ax=29: LET
   ay=16: LET dead=0: LET props=3
3090 GOSUB 5000
3150 PAPER 0: BORDER 0: CLS
3160 FOR x=0 TO 28: PRINT AT 3,x; INK 0;
   PAPER 6;"□";AT 0,x;"□": NEXT x:
   GOSUB 6000
3170 POKE 23607,60: PRINT AT 0,0; INK 0;
   PAPER 6;"L=";level;"□□B=";bl;
   "□□";"SC=";score;AT 0,20;"HI=";
   hiscore
3180 POKE 23607,252
3190 FOR y=5 TO 21: PRINT AT y,30; INK
   6;"kl": NEXT y
3200 POKE 23607,252
3210 GOSUB 4000
3220 GOSUB 4200
3240 RETURN
```

```
3000 SC=0:LV=1
3010 MY=15
3020 BL=15+5*LV:AX=29:AY=16:
   DD=0:PP=3
3090 GOSUB 5000
3150 PMODE4,1:COLOR 0,1:PCLS:SCREEN1,1
3160 FOR X=0 TO 28:PUT(X*8,24)-
   (X*8+7,31),BL,PSET:PUT(X*8,0)-
   (X*8+7,7),BL,PSET:NEXTX:GOSUB 6000
   3165 DRAW "S4;BM2,2;C1;D4R3BR2BU1
      R2BU1L2;BM48,2;D4R4U2L4R3U2L3BR6
      BD2R2BD1L2;BM100,2;L4D2R4D2L4BR6
      NR4U4R4;BM160,2;D4U2R4D2U4BR2R4L2
      D4L2R4;S2"
3170 DRAW "C1;BM14,2;":NU=LV:GOSUB
   1650:DRAW "BM58,2;":NU=BL:GOSUB
   1650:DRAW "BM114,2;":NU=SC:GOSUB
   1650:DRAW "BM178,2;":NU=HS:
   GOSUB 1650
3190 FOR Y=5 TO 21:PUT(240,Y*8)-(255,
   Y*8+7),KL,PSET:NEXTY
3210 GOSUB 4000
3220 GOSUB 4200
3240 RETURN
```



Lines 3000, 3010 and 3020 set the score to zero and level to one, and initialize a series of variables. Line 3150 sets the screen colours, and Lines 3160 and 3170 display the score and other information.

Something should be said about the POKEs in Lines 3170 and 3180 of the Spectrum program. Memory location 23607 stores a pointer to the character set. Usually this location has the value 60, which points to the normal ROM character set. In this program, though, the characters have been placed in an area which is referenced by POKEing 23607 with 252 instead. Line 3180 POKEs 23607 with 252 so the program can use the lower case letters as graphics, but still allows you use of numerals and upper case letters.

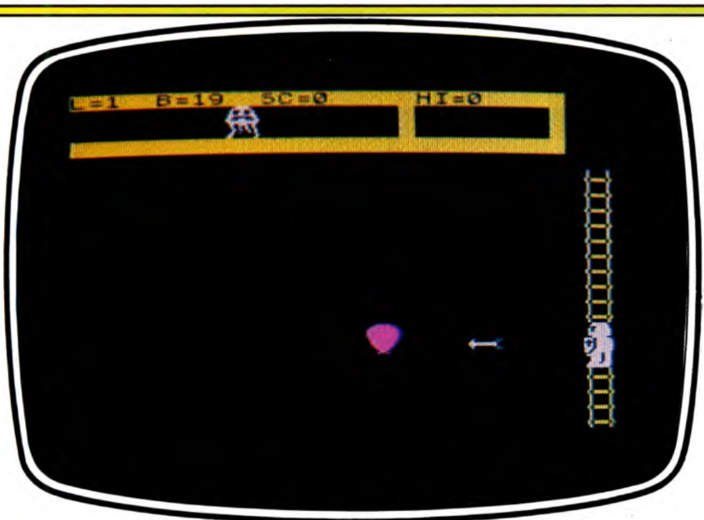
The program reserves an area of memory, using CLEAR, and plants the UDGs there. The CLEAR has to be executed as part of the main program because of the way it affects the execution of the subroutines. This will be done in the second part of this game—if you want to run this part of the program before then, type CLEAR 65287 first.

Line 3190 draws the ladder, and the two subroutine calls draw Freddy and the spider.

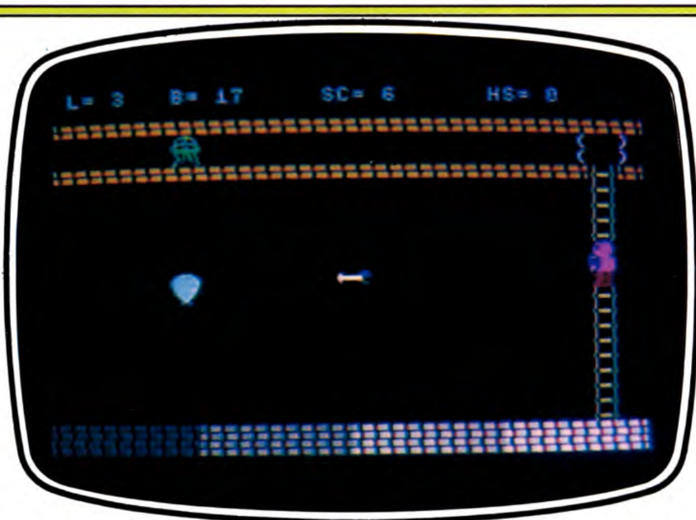
FREDDY AND THE ARROW

```
4000 INK 7: PRINT AT my,30;"uv";AT my+1,
   30;"wx";AT my+2,30;"yz": IF ax=29
   THEN PRINT AT ay,ax;"e";
4010 RETURN
4110 INK 7: PRINT AT ay,ax;"ef": RETURN
```

```
4000 PUT(240,MY*8)-(255,MY*8+23),UZ,
   PSET:IF AX=29 THEN PUT(AX*8,AY*8)-
```



Freddy and the spider from Mars shown in full flight on the Spectrum



More detailed colour graphics in the Commodore version of the game

```
    (AX*8+7,AY*8+7),E,PSET
4Ø1Ø RETURN
411Ø PUT(AX*8,AY*8)-(AX*8+15,AY*8+7),
    EF,PSET:RETURN
```

This routine simply draws Freddy on the ladder, and the arrow. Freddy's position is determined by the value of MY (or my). Similarly, Line 411Ø will draw the arrow at any position on the screen, according to the values of AX and AY (or ay and ax).

THE MARTIAN SPIDER

```
42ØØ IF s(picture)=1 THEN GOTO 425Ø
421Ø PRINT AT s(ypos),s(xpos);"mn";AT
    s(ypos)+1,s(xpos);"op": RETURN
425Ø PRINT AT s(ypos),s(xpos);"qr";AT
    s(ypos)+1,s(xpos);"st": RETURN
```

```
42ØØ X2=S(XP)*8:Y2=S(YP)*8:IF S(PI)=1
    THEN 425Ø
421Ø PUT(X2,Y2)-(X2+15,Y2+15),MP,
    PSET:RETURN
425Ø PUT(X2,Y2)-(X2+15,Y2+15),QT,
    PSET:RETURN
```

This routine is very similar to the other printing routines, but it has to be slightly more subtle because there are two images to play with. Both images are printed at the same place on the screen, one after the other, so the spider's legs appear to move.

THE BALLOON GOES UP

```
43ØØ PRINT AT b(ypos),b(xpos); BRIGHT 1;
    INK b(colour);"ab";AT b(ypos)+1,b(xpos);
    "cd": RETURN
5ØØØ LET range=INT (RND*6)
5Ø1Ø LET b(xpos)=(4*range)+INT (RND*4)
5Ø2Ø LET b(ypos)=2Ø
5Ø3Ø LET b(maxcount)=5-level
5Ø4Ø LET b(count)=1
5Ø5Ø LET b(colour)=INT (RND*5)+3
5Ø6Ø LET b(points)=1Ø-range
5Ø7Ø RETURN
```

```
43ØØ X2=B(XP)*8:Y2=B(YP)*8:PUT(X2,
    Y2)-(X2+15,Y2+23),AD,PSET:RETURN
5ØØØ RG=RND(6)-1
5Ø1Ø B(XP)=(4*RG)+RND(4)-1
5Ø2Ø B(YP)=2Ø
5Ø3Ø B(MC)=5-LV
5Ø4Ø B(CT)=1
5Ø6Ø B(PO)=1Ø-RG
5Ø7Ø RETURN
```

Line 43ØØ simply draws the balloons on screen.

The remaining lines inflate a new balloon once the previous one has been popped by an arrow, or by reaching the beast's cage. Balloons may appear at any one of six places at the bottom of the screen, and then float vertically upwards. MAXCOUNT or (maxcount) determines how often the balloon is moved and hence its apparent speed, and is related to the level the game has reached. The colour of the balloon is chosen in the Spectrum version, and finally the points value of the balloon is set according to how close it is to Freddy's ladder.

THE DOORS

```
6ØØØ IF level<>1 THEN POKE 236Ø7,6Ø:
    PRINT AT s(ypos),s(xpos);"□□";AT s(ypos)
    +1,s(xpos);"□□": POKE 236Ø7,252
6Ø1Ø FOR x=1Ø TO 3Ø STEP 9
6Ø2Ø PRINT INK 6;AT 1,x;"?";AT 2,x;"?"
6Ø3Ø NEXT x
6Ø4Ø LET s(xpos)=1: LET s(ypos)=1: LET
    s(xinc)=1: LET s(yinc)=Ø: LET
    s(count)=4: LET s(maxcount)=4: LET
    s(picture)=1
6Ø5Ø RETURN
```

```
6ØØØ IF LV<>1 THEN X2=S(XP)*8:Y2=S
    (YP)*8:PUT(X2,Y2)-(X2+15,Y2+15),
    SP,PSET
6Ø1Ø FOR X=1Ø TO 3Ø STEP 9
6Ø2Ø PUT(X*8,8)-(X*8+7,15),BL,PSET:PUT
    (X*8,16)-(X*8+7,23),BL,PSET
6Ø3Ø NEXT X
6Ø4Ø S(XP)=1:S(YP)=1:S(XI)=1:S(YI)=Ø:
    S(CT)=4:S(MC)=4:S(PI)=1
6Ø5Ø RETURN
```

Three doors are drawn in the Spider's cage to keep it imprisoned. Spectrum owners note that the ? character is in fact a blob, obtained from the 8 key while in graphics mode.

The Commodore program is different from the rest, so slightly different aspects of the program will be covered here.

GRAPHICS INITIALIZATION

```
1Ø POKE 51,255:POKE 52,47:POKE 55,255:
    POKE 56,47:CLR:PRINT"♥PLEASE
    WAIT..."
15 GOSUB 1ØØØ
999 GOTO999
1ØØØ POKE 56334,Ø:POKE 1,51:FOR I=Ø TO
    64*8-1:POKE 12288+I,PEEK(53248+I):
    NEXT I
```

```
1Ø1Ø POKE 1,55:POKE 56334,1
1Ø2Ø FOR Z=13312 TO 13527:READ X:
    POKE Z,X:NEXT Z:RETURN
1Ø5Ø DATA15,63,127,255,255,255,
    255,127
1Ø6Ø DATA24Ø,252,254,255,255,255,
    255,254
1Ø7Ø DATA127,63,63,31,15,7,3,6
1Ø8Ø DATA254,252,252,248,24Ø,224,
    192,96
1Ø9Ø DATA32,96,255,255,96,32,Ø,Ø
11ØØ DATA5,1Ø,252,252,1Ø,5,Ø,Ø
111Ø DATA1,64,17,4Ø,16,Ø,Ø,161
112Ø DATA128,2,136,2Ø,8,Ø,Ø,133
113Ø DATA161,Ø,Ø,16,4Ø,17,64,1
114Ø DATA133,Ø,Ø,8,2Ø,136,2,128
115Ø DATA48,48,48,48,111,111,48,48
116Ø DATA12,12,12,12,246,246,12,12
117Ø DATA7,31,49,57,127,112,237,255
118Ø DATA224,248,14Ø,2Ø4,254,14,
    183,255
119Ø DATA127,59,51,99,115,35,6,12
12ØØ DATA254,1Ø8,1Ø2,51,49,25,24,48
121Ø DATA7,31,49,51,127,112,235,255
122Ø DATA224,248,14Ø,156,254,14,
    215,255
123Ø DATA127,51,5Ø,27,25,5Ø,112,224
124Ø DATA254,2Ø4,1Ø8,1Ø2,54,22,3,6
125Ø DATA5,31,19,55,55,63,63,15
126Ø DATA24Ø,248,248,252,252,252
    252,24Ø
127Ø DATA251,219,139,219,219,251
    247,239
128Ø DATA252,254,254,254,254,254
    254,252
129Ø DATA95,127,31,31,31,63,127,127
13ØØ DATA188,188,188,188,188,124,
    252,248
131Ø DATA239,239,239,Ø,247,247,
    247,Ø
```

This routine contains all the DATA needed for setting up the graphics. UDGs are set up for Freddy, the Spider, the ladder, and the platforms at the top and bottom of the screen by POKEing into memory, after Line 1Ø clears sufficient space for them.

ERECTING THE SCREEN

```
2Ø POKE 53272,28: GOSUB 2ØØØ
2ØØØ POKE 5328Ø,Ø:POKE 53281,Ø:PRINT
    "■■■L=■Ø■■■B=■2Ø■
    ■■■■SC=■Ø■■■■■■
    HS="HS"■"
2Ø1Ø FOR Z=1TO 2:PRINT"■■ZZZZZZZZZ
    ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ■
    ■■";:NEXT Z
2Ø15 PRINT"■■■■";
2Ø2Ø FOR Z=1TO 2:PRINT"■
    ■■■■■■■■■■■■■)
    ■■■■■■■■■■■■■)
    ■■■■■■■■■■■■■)
    ■■)":NEXT Z
2Ø3Ø PRINT TAB(36)"π■JK":PRINT
    "■";:FOR Z=1 TO 16:PRINT SPC(36)
    "π■JK":NEXT Z
2Ø4Ø FORZ=1TO2:PRINT"■■ZZZZZZZZZZ
    ■ZZZZZZZZZZ■ZZZZZZZZZZ■ZZZZZZZ
    ZZZ■";
2Ø5Ø NEXT Z:RETURN
```

Line 2Ø switches to the new character set and then jumps to the routine at Line 2ØØØ. The screen is drawn, including the score and high score positions. Line 2Ø1Ø draws the spider's cage, 2Ø3Ø the ladder, and 2Ø4Ø the ground.

Part one sees the graphics set up and the game initialized.

GRAPHICS INITIALIZATION

```
67Ø DEFPROCINITIALIZE
68Ø DIM B%(6),S%(7)
69Ø VDU 23,1,Ø;Ø;Ø;Ø;
7ØØ XPOS%=1:YPOS%=2:CLR%=3:
    POINTS%=4:CNT%=5:MAXCOUNT%=6
71Ø VDU 19,1,2,Ø,Ø,Ø,19,2,5,Ø,Ø,Ø
72Ø XINC%=3:YINC%=4:PICTURE%=7
73Ø FOR I%=Ø TO (27*8)-1:READJ%:?
    (&CØØ+I%)=J%:NEXTI%
74Ø DATA 15,63,127,255,255,255,
    255,127
75Ø DATA 24Ø,252,254,255,255,255,
    255,254
76Ø DATA 127,63,63,31,15,7,3,6
77Ø DATA 254,252,252,248,24Ø,224,
    192,96
78Ø DATA 32,96,255,255,96,32,Ø,Ø
79Ø DATA 5,1Ø,252,252,1Ø,5,Ø,Ø
8ØØ DATA 1,64,17,4Ø,16,Ø,Ø,161
81Ø DATA 128,2,136,2Ø,8,Ø,Ø,133
82Ø DATA 161,Ø,Ø,16,4Ø,17,64,1
83Ø DATA 133,Ø,Ø,8,2Ø,136,2,128
84Ø DATA 48,48,48,48,111,111,48,48
85Ø DATA 12,12,12,12,246,246,12,12
86Ø DATA 7,31,49,57,127,112,237,255
87Ø DATA 224,248,14Ø,2Ø4,254,14,
    183,255
88Ø DATA 127,59,51,99,115,35,6,12
89Ø DATA 254,1Ø8,1Ø2,51,49,25,24,48
9ØØ DATA 7,31,49,51,127,112,235,255
91Ø DATA 224,248,14Ø,156,254,14,
    215,255
92Ø DATA 127,51,5Ø,27,25,5Ø,112,224
93Ø DATA 254,2Ø4,1Ø8,1Ø2,54,22,3,6
94Ø DATA 15,31,19,55,55,63,63,15
95Ø DATA 24Ø,248,248,252,252,252,
    252,24Ø
96Ø DATA 251,219,139,219,219,251,
    247,239
97Ø DATA 252,254,254,254,254,254,
    254,252
98Ø DATA 95,127,31,31,31,63,127,127
99Ø DATA 188,188,188,188,188,124,
    252,248
1ØØØ DATA 255,255,255,255,255,255,
    255,255
1Ø1Ø HISCORE%=Ø:ENDPROC
```

Arrays B and S hold DATA concerned with the balloon and the spider. Integer arrays are DIMensioned in Line 68Ø to economize on memory. Line 69Ø switches off the cursor.

Line 7ØØ initializes a series of variables which are used to access the DATA in the arrays when setting up the UDGs. Finally, Line 1Ø1Ø sets the high score to zero, and is placed here as it occurs only once in the program.

GAME INITIALIZATION

```
1Ø2Ø DEFPROCGAMEINIT
1Ø3Ø SCORE%=Ø:LEVEL%=1
1Ø4Ø MY%=15:BL%=15+5*LEVEL%:
    AX%=29:AY%=16:DEAD%=Ø:
    PROPS%=3
1Ø5Ø PROCMAKEBALLOON
1Ø6Ø COLOUR2:FORX%=Ø TO 28:PRINT
    TAB(X%,3);CHR$(154);TAB(X%,Ø);CHR$
    (154):NEXTX%
1Ø7Ø PROCDRAWDOORS
1Ø8Ø COLOUR13Ø:COLOURØ:PRINT TAB
    (Ø,Ø);"L=";LEVEL%;"□□B=";BL%;
    "□□SC=";SCORE%
1Ø9Ø PRINT TAB(2Ø,Ø);'HI=";HISCORE%:
    COLOUR128:COLOUR2
11ØØ COLOUR2:FOR Y%=5 TO 21:PRINT
    TAB(3Ø,Y%);CHR$(138);CHR$(139):NEXT
    Y%
111Ø PROCDRAWMAN
112Ø PROCDRAWSPIDER
113Ø ENDPROC
```

In this PROCedure, the score, the level, and a whole series of variables are set.

Line 1Ø5Ø calls PROCMAKEBALLOON, which will be explained a little later, but draws a new balloon on screen.

The next section of the program sets up the screen. Line 1Ø6Ø draws the platform along the top of the screen, and Line 1Ø7Ø draws the props. Game information is displayed by Lines 1Ø8Ø and 1Ø9Ø. Freddy's ladder is drawn by Line 11ØØ, and Freddy and the spider by calling the appropriate PROCedures.

FREDDY AND THE ARROW

```
114Ø DEFPROCDRAWMAN
115Ø COLOUR 3
116Ø PRINT TAB(3Ø,MY%)CHR$(148)CHR$
    (149) TAB(3Ø,MY%+1)CHR$(15Ø)CHR$
    (151) TAB(3Ø,MY%+2)CHR$(152)CHR$
    (153)
117Ø IF AX%=29 THEN PRINT TAB(29,
```

```
    MY%+1)CHR$(132)
118Ø ENDPROC
119Ø DEFPROCDRAWARROW
12ØØ COLOUR:PRINT TAB(AX%,AY%)CHR$
    (132)CHR$(133)
121Ø ENDPROC
```

PROCDRAWMAN simply draws Freddy on the ladder. Freddy's position is determined by the value of MY%. In a similar way, PRACDRAWARROW will draw the arrow at any position on the screen, according to the values of AX% and AY%.

THE MARTIAN SPIDER

```
122Ø DEFPROCDRAWSPIDER
123Ø COLOUR1
124Ø IF S%(PICTURE%)=1 THEN 126Ø
125Ø PRINT TAB(S%(XPOS%),S%(YPOS%))
    CHR$(14Ø)CHR$(141)TAB(S%(XPOS%),
    S%(YPOS%)+1)CHR$(142)CHR$(143):
    ENDPROC
126Ø PRINT TAB(S%(XPOS%),S%(YPOS%))
    CHR$(144)CHR$(145)TAB(S%(XPOS%),
    S%(YPOS%)+1)CHR$(146)CHR$(147):
    ENDPROC
```

This PROCedure is very similar to the other printing PROCedures, but it has to be slightly more subtle because there are two images to play with. Both images are printed at the same place on the screen, one after the other, so the spider's eight hairy legs appear to move as it scuttles backwards and forwards in its cage.

THE BALLOON GOES UP

```
127Ø DEFPROCDRAWBALLOON
128Ø COLOUR B%(CLR%):PRINT TAB(B%
    (XPOS%),B%(YPOS%))CHR$(128)CHR$
    (129)TAB(B%(XPOS%),B%(YPOS%)+1)
    CHR$(13Ø)CHR$(131):ENDPROC
129Ø DEFPROCMAKEBALLOON
13ØØ RANGE%=RND(1)*6
131Ø B%(XPOS%)=(4*RANGE%)+RND
    (1)*4
132Ø B%(YPOS%)=2Ø
133Ø B%(MAXCOUNT%)=5-LEVEL%
134Ø B%(CNT%)=1
135Ø B%(CLR%)=RND(3)
136Ø B%(POINTS%)=1Ø-RANGE%
137Ø ENDPROC
```

Line 128Ø in PROCDRAWBALLOON simply draws the balloons on screen, according to the values of XPOS% and YPOS%.

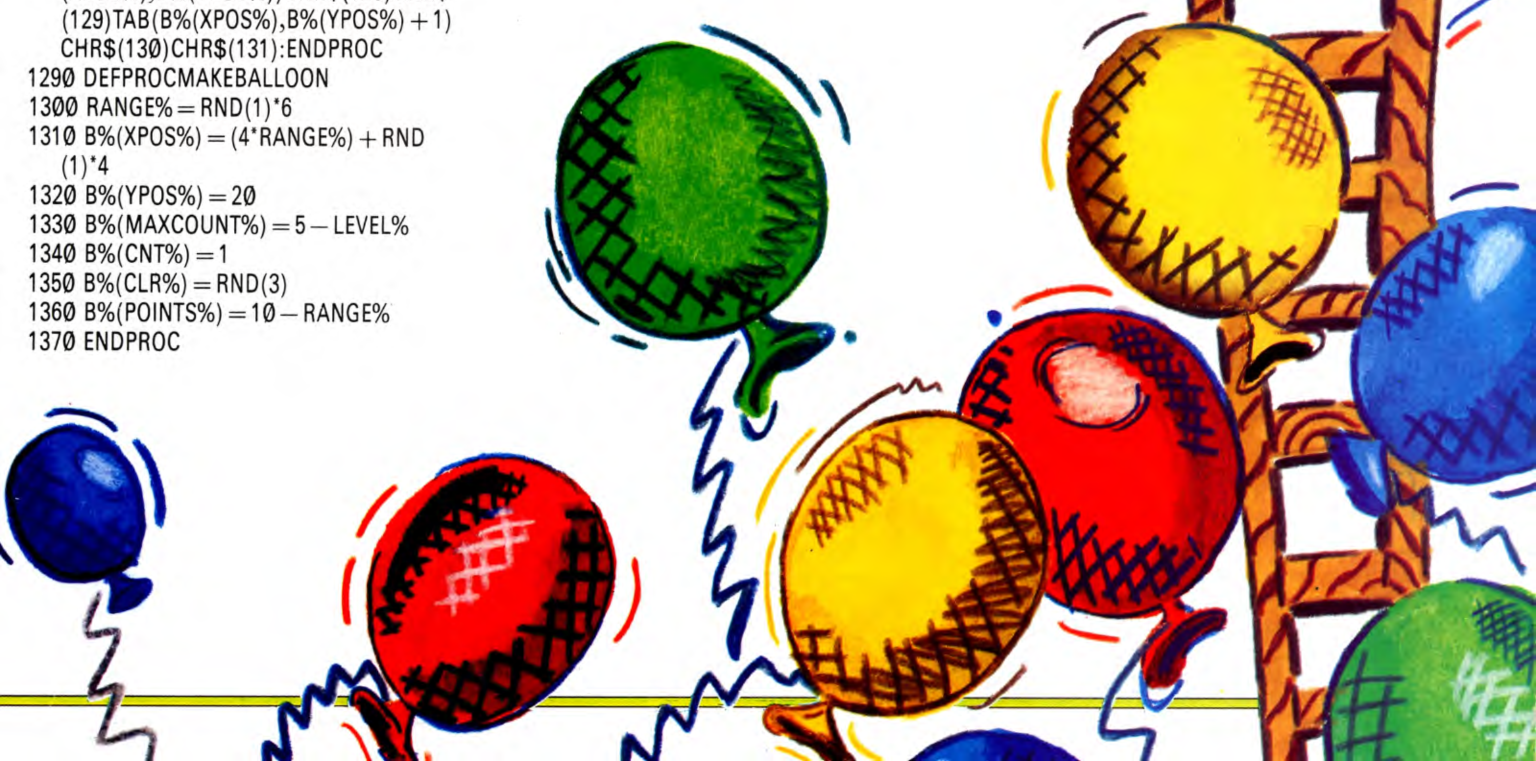
PROCMAKEBALLOON—Lines 129Ø to 137Ø—inflate a new balloon once the previous one has been popped by an arrow, or by reaching the beast's cage. Balloons may appear at any one of six places at the bottom of the screen, and then float vertically upwards. MAXCOUNT% determines how often the balloon is moved, and is related to the level the game has reached. And finally the points value of the balloon is set according to how close it is to Freddy's ladder.

THE DOORS

```
138Ø DEFPROCDRAWDOORS
139Ø COLOUR2:IF LEVEL%<>1 THEN
    PRINT TAB(S%(XPOS%),S%(YPOS%))
    "□"TAB(S%(XPOS%),S%(YPOS%)+1)"□"
14ØØ FOR X%=1Ø TO 3Ø STEP 9
141Ø PRINT TAB(X%,1)CHR$(154)TAB(X%,2)
    CHR$(154)
142Ø NEXT X%
143Ø S%(XPOS%)=1:S%(YPOS%)=1:
    S%(XINC%)=1:S%(YINC%)=Ø:S%
    (PICTURE%)=1
144Ø ENDPROC
```

Three locks are drawn in the spider's cage by PROCDRAWDOORS to keep it imprisoned.





WORKING WITH SPREADSHEETS

Here is the last part of the program. Type it in now and you can start to fill in the details and use the spreadsheet. The instructions explain exactly what to do

If you add these remaining lines you will have a complete, working spreadsheet program, so LOAD in the first parts now and enter the last section listed below. A few examples were given in the earlier parts showing how you might set up a spreadsheet. Now you can find out the details of how to use the individual programs—which keys do what and so on. The general instructions that follow are the same for all the programs but there are a few differences in the commands and these are listed in the individual sections for each machine at the end.

ENTERING EQUATIONS

It is the ability to enter equations into the spreadsheet that makes it so versatile. Don't be put off by the word 'equations' though, as these use no more than simple arithmetic such as adding columns together, totalling a whole row or working out percentages for VAT, discount, depreciation and so on.

The operators used are plus, minus, multiply, divide, percent, row and column totals: +, −, *, /, %, & and $. Equations are entered by specifying the name of the first cell, then the name of the second cell and then the operator. A few examples will help. Entering A1B1+ in cell C1 adds the contents of A1 and B1 and displays it in cell C1. The equation A1A1Ø$ sums all the contents in column A from row 1 to row 1Ø. Similarly, A6F6& sums all the entries in row 6 from column A to column F. Finally, A5B2% is worked out as A5*B2/1ØØ and gives B2 percent of A5. The equations are entered into the cell in which you want the answer to appear. The Acorn, Dragon and Tandy programs also allow you to add a number after the equation to specify the number of decimal places that will appear in the answer.

COPYING

All the programs allow you to copy or *replicate* the contents of a cell. This saves you having to type in the same thing over and over again when you want a value or equation repeated in several cells. You can choose to copy something either *absolutely* or *relatively*. An absolute copy produces an exact replica of the cell anywhere else you choose—either in a single cell or along a section of a row or column. The relative copy is used specially for equations so that the equation is altered depending on the name of the row or column it is copied into.

For example, if you have the equation A1B1* in cell C1, you might want every cell in column C to multiply the numbers in the previous two columns together. So in C2 you'd have A2B2* and so on. The relative copy does this for you. The procedure is as follows. First press the copy key (see below for the relevant key for each computer), then choose R for relative and enter the name of the cell you want copied (or on the Dragon and Tandy move the cursor to the correct cell). Then choose whether you want the cell copied along a row or column—press C in the case of this example. Finally enter the name of the start cell—say, C1—and the end cell—say, C1Ø. The program then fills in the whole column for you. Try copying equations along rows as well, and try an absolute copy of equations and values.

UNCHANGING CONSTANTS

It is often necessary to enter constants into equations. These are likely to be numbers such as 15 for percentage VAT calculations, perhaps 5 or 1Ø for percentage discounts, 3Ø for tax and so on. Numbers cannot be entered directly into an equation—which is only based on the contents of two cells—and the method of putting them in varies for the different computers.

On the Spectrum the constant has to be preceded by the letter Z, so A1Z15% works out VAT payable on the figure in A1. On the Commodore and Acorn computers the constants have to be entered into cells in column X. So in the last example, number 15 would be put into cell X1 and VAT would be worked out as A1X1%. The Dragon and Tandy program is much the same as this except the constants are entered into column Z.

The reason for having a special method for dealing with constants is that these numbers or cells must stay the same even when copied relatively. Reserving a special letter or a special column for the constants ensures that these are treated separately.

USING THE PROGRAM

As mentioned, the specific details of how values, labels and equations are entered into the computer and how all the operations are performed varies between the four computers so these are listed separately below.

S

The Spectrum program takes a little while to set up so you'll see a blank screen for a while after you type RUN. The screen starts off in the values mode which means you can insert numbers or labels. Press E to swap to the equations mode and V to swap back again. To insert anything into a cell use the cursor key to move to the correct cell, then press I (for Insert) and type in the entry.

Equations are displayed on a yellow background, values on black and labels on blue. A label is anything entered into the values screen that starts with an alphabetic character. The colours show through each sheet so you can see which cells contain equations even when you are looking at the values screen, and vice versa.

Only four columns and ten lines fit on the screen at any one time but you can move to other parts of the spreadsheet by pressing the cursor keys along with [SYMBOL SHIFT].

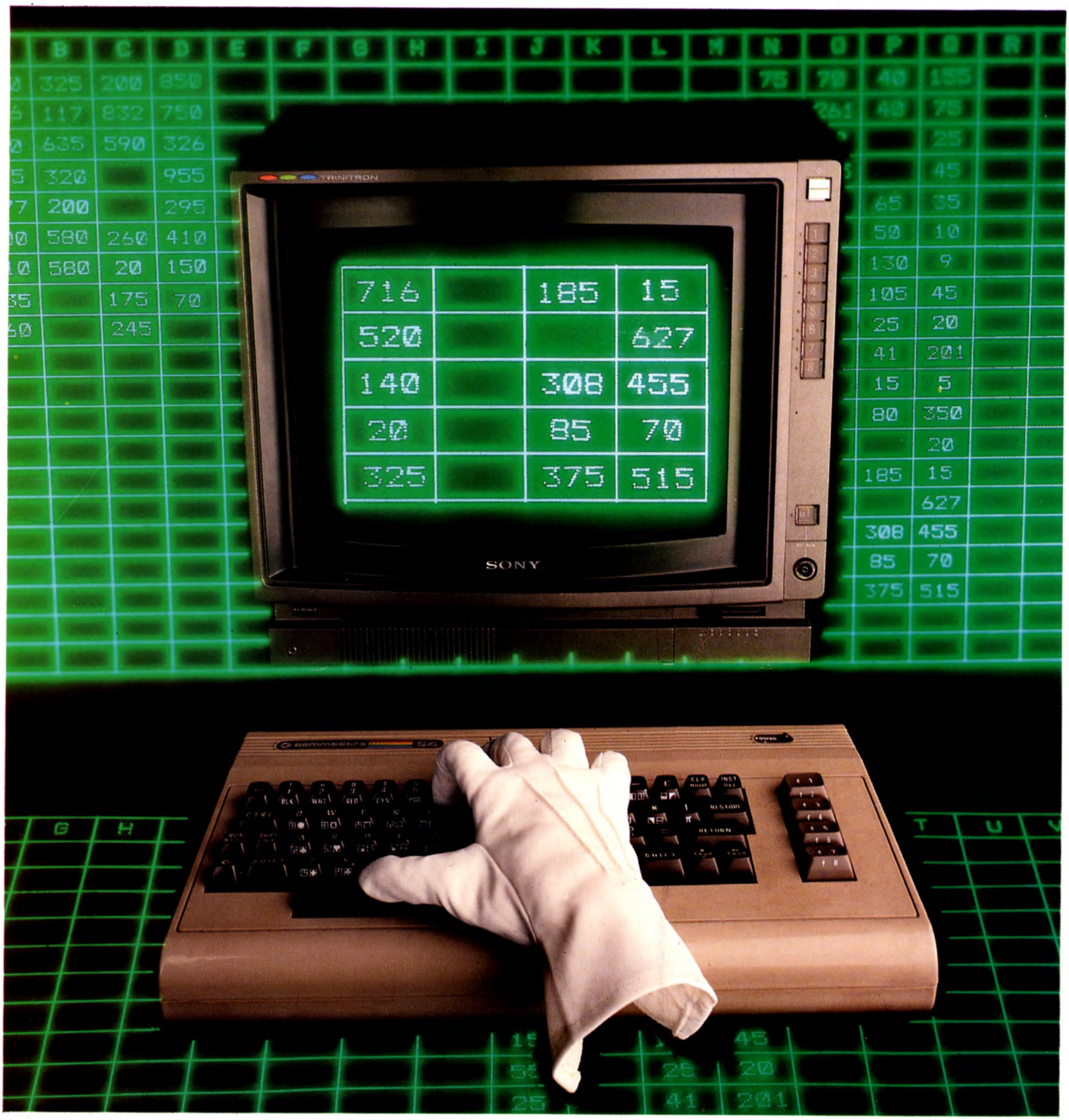
Try inserting a few values and labels, then swap to the equations screen and try adding cells together, working out percentages and summing columns and rows. To see the results of the calculations swap back to the values screen, press C and [SYMBOL SHIFT] to calculate the values. If any numbers are too big to fit into a cell, the rightmost digits that overflow will be cut off and the remaining number will flash as a warning.

Now try copying some equations and values using the Z key. The program will prompt you to enter all the necessary details.

Finally, here is a list of all the commands used by the program:

V	display values screen
E	display equations screen
I	insert an entry into a cell
I	followed by [ENTER] delete a value or label
I	followed by # delete an equation

- GENERAL INSTRUCTIONS
- HOW TO ENTER THE EQUATIONS
- ABSOLUTE AND RELATIVE COPYING
- USING CONSTANTS IN THE EQUATIONS
- SPECIAL INSTRUCTIONS FOR EACH COMPUTER
- FINISHING OFF THE PROGRAM



C plus [SYMBOL SHIFT] calculate the values of any equations
[SPACE] plus [SYMBOL SHIFT] abort calculation
Z copy a cell
S plus [SYMBOL SHIFT] save data to tape
J plus [SYMBOL SHIFT] load data from tape
P print screen on ZX or Alphacom printer
Cursor keys move cursor
Cursor keys plus [SYMBOL SHIFT] move window

When you RUN the program you'll see four columns and 15 rows of the spreadsheet, but you can move to other parts of the sheet using the cursor keys or the function key [f5]. Six function keys are used to control the program and these are listed at the bottom of the screen. Here's a description of what each of the keys do:

[f1] swap between values and equations screen
[f2] alter a cell
[f3] copy a cell
[f4] calculate results of equations
[f5] move window
[f6] save data and exit program

To make an entry, first make sure you are on the correct screen for the type of entry you want to make, then press [f2] and type in the value, label or equation. A label is anything entered on the values screen that starts with an alphabetic character.

To copy a cell, press [f3] and you will be prompted to enter all the necessary details. Press [f4] to calculate the equations. If any number is too big to fit into a cell then the words TOO BIG will be printed instead.

Use [f5] to move to a different part of the screen by specifying the cell you want to see in the top left-hand corner.

Try entering some values and equations into the sheet and practise copying, both absolutely and relatively, and calculating the results of equations.

The Acorn program uses the function keys to control the program and when you RUN the program these are listed at the bottom of the screen underneath the spreadsheet. Only a small part of the spreadsheet is visible at any time but you can move the 'paper' under the window with the cursor keys, or with [f4] for a large move.

Key [f0] swaps between values and equations screens and [f1] lets you make an entry into a cell. Key [f3] calculates the values of any equations and displays the results on the values screen. If a number is too big to fit into a cell then the words TOO BIG are printed instead.

The Acorn program allows you to specify the number of decimal places of any calculated value. The number can be entered at the end of an equation, so A1B1*2 will display the answer to two decimal places—useful when dealing with quantities of money. It is also useful if a number is likely to be too big as specifying zero decimal places may mean it will fit on the screen.

Here is a list of the keys used:

[f0] swap between values and equations screen
[f1] alter cell
[f2] copy a cell
[f3] calculate equations
[f4] move window
[TAB] save data and exit program
[ESCAPE] escape from current operation without losing data

When you RUN the program you'll see just the top left-hand corner of the spreadsheet but you can move to any other part of the sheet with the cursor keys, or by pressing G for a large move. The V and E keys swap between the values screen and the equations screen. You can enter values or numbers on either screen but the results of calculations are only displayed on the values screen. To insert anything into a cell move the cursor to the correct cell then press I and type in your entry.

Equations are entered as described earlier except that you can add a number to the end of the equation to specify the number of decimal places in the result. For example, A1B1/2 will display the result of A1 divided by B1 to two decimal places, which is ideal for dealing with quantities of money. If the result of any calculation is too big to fit into a cell then the overflow message <OV> will be printed instead. Specifying zero decimal places in the equation that works out that result may mean that the answer will fit in the cell.

Here is a list of all the keys used by the program:

V display value screen
E display equation screen
I insert label, value or equation
C copy a cell
S save data to tape
L load data from tape
Q quit program
Cursor keys move cursor and shift 'paper'

```
910 IF z=3 THEN LET v(1)=(CODE
    s$(1))-64: LET v(2)=VAL s$(2): LET
    v(3)=(CODE s$(3))-64: LET v(4)=VAL
    s$(4 TO 5): LET o$=s$(6): RETURN
920 IF z=4 THEN LET v(1)=(CODE
    s$(1))-64: LET v(2)=VAL s$(2 TO 3):
    LET v(3)=(CODE s$(4))-64: LET
    v(4)=VAL s$(5 TO 6): LET o$=s$(7):
    RETURN
930 IF z=5 THEN LET v(1)=(CODE
    s$(1))-64: LET v(2)=VAL s$(2): LET
    v(3)=(CODE s$(3))-64: LET v(4)=VAL
    s$(4): LET o$=s$(5): RETURN
940 IF z=6 THEN LET v(1)=(CODE
    s$(1))-64: LET v(2)=VAL s$(2 TO 3):
    LET v(3)=(CODE s$(4))-64: LET
    v(4)=VAL s$(5): LET o$=s$(6): RETURN
950 IF z=7 THEN LET v(1)=(CODE
    s$(1))-64: LET v(2)=VAL s$(2): LET
    v(3)=(CODE s$ (3))-64: LET v(4)=VAL
    s$(4 TO 5): LET o$=s$(6): RETURN
960 IF z=8 THEN LET v(1)=(CODE
    s$(1))-64: LET v(2)=VAL s$(2 TO 3):
    LET v(3)=(CODE s$(4))-64: LET
    v(4)=VAL s$(5 TO 6): LET o$=s$(7):
    RETURN
970 IF z=9 THEN LET v(1)=(CODE
    s$(1))-64: LET v(2)=VAL s$(2): LET
    v(3)=(CODE s$(3))-64: LET v(4)=VAL
    s$(4 TO 6): LET o$=s$(7): RETURN
980 IF z=10 THEN LET v(1)=(CODE
    s$(1))-64: LET v(2)=VAL s$(2 TO 3):
    LET v(3)=(CODE s$(4))-64: LET
    v(4)=VAL s$(5 TO 7): LET o$=s$(8):
    RETURN
990 LET v(1)=(CODE s$(1))-64: LET
    v(2)=VAL s$(2): LET v(3)=(CODE
    s$(3))-64: LET v(4)=VAL s$(4 TO 7):
    LET o$=s$(8): RETURN
1000 RETURN
1010 IF d$(v(2),v(1),1)>"9" THEN LET
    t=0: RETURN
1020 IF v(3)=26 THEN LET b=v(4): LET
    a=VAL d$(v(2),v(1), TO 8): GOTO 1050
1030 IF d$(v(4),v(3),1)>"9" THEN LET
    t=0: RETURN
1040 LET a=VAL d$(v(2),v(1), TO 8):
    LET b=VAL d$(v(4),v(3), TO 8)
1050 IF o$="+" THEN LET t=a+b:
    RETURN
1060 IF o$="-" THEN LET t=a-b:
    RETURN
1070 IF o$="/" AND b=0 THEN LET t=0:
    RETURN
1080 IF o$="/" THEN LET t=a/b: RETURN
1090 IF o$="*" THEN LET t=a*b: RETURN
1100 IF o$="%" THEN LET t=(a*b)/100:
    RETURN
1110 IF o$="$" THEN LET t=0: FOR
```

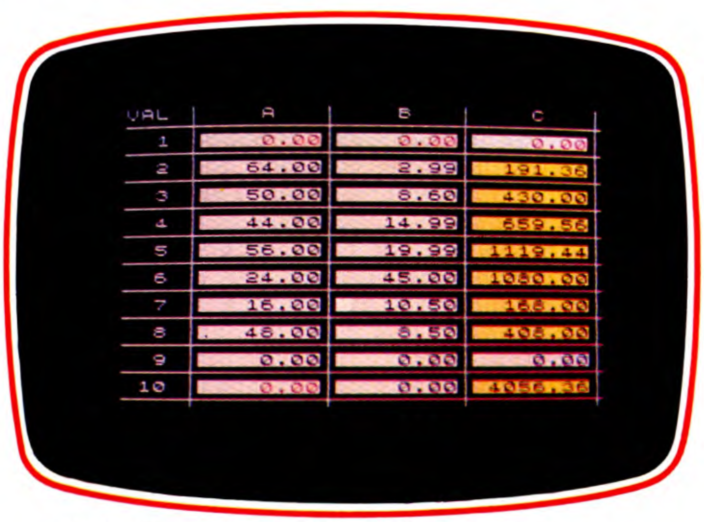
```
s=v(2) TO v(4): LET t=t+VAL d$(s,v(1),
   TO 8): NEXT s: RETURN
1120 IF o$="&" THEN LET t=Ø: FOR
   s=v(1) TO v(3): LET t=t+VAL d$(v(2),s,
   TO 8): NEXT s: RETURN
1130 RETURN
1140 IF z=1 THEN IF VAL a$(2)=Ø OR VAL
   a$(4)=Ø THEN LET f=1: RETURN
1150 IF z=2 THEN IF VAL a$(2 TO 3)>30
   OR VAL a$(2 TO 3)=Ø OR VAL a$(5)=Ø
   THEN LET f=1: RETURN
1160 IF z=3 THEN IF VAL a$(4 TO 5)>30
   OR VAL a$(4 TO 5)=Ø OR VAL a$(2)=Ø
   THEN LET f=1: RETURN
1170 IF z=4 THEN IF VAL a$(2 TO 3)>30
   OR VAL a$(5 TO 6)>30 OR VAL a$(2 TO
   3)=Ø OR VAL a$(5 TO 6)=Ø THEN LET
   f=1: RETURN
1180 IF z=5 OR z=7 OR z=9 OR z=11
   THEN IF VAL a$(2)=Ø THEN LET f=1:
   RETURN
1190 IF z=6 OR z=8 OR z=10 THEN IF
   VAL a$(2 TO 3)=Ø OR VAL a$(2 TO
   3)>30 THEN LET f=1: RETURN
1200 LET s$="□□□□□□□□": FOR
   g=1 TO c: LET s$(g)=a$(g): NEXT g:
   GOSUB 890: IF o$="&" AND
   (v(1)>v(3) OR v(2)<>v(4)) THEN LET
   f=1: RETURN
1210 IF o$="$" AND (v(1)<>v(3) OR
   v(2)>v(4)) THEN LET f=1: RETURN
1220 LET f=Ø: RETURN
1230 LET g$=d$ (y+wy,x+wx,17)
1240 PRINT FLASH Ø+(g$="5" AND
   t$="VAL"); INK Ø+(2 AND g$="4");
   PAPER 7-(2 AND g$="2")-(g$="1"
   OR g$="5");AT
   y*2+2,x*9+5;(d$(y+wy,x+wx, TO 8)
   AND t$="VAL"); (d$(y+wy,x+wx,9 TO
   16) AND t$="EQU"): RETURN
1250 BEEP .4,10
1260 LET s$="□□□□□□□□": LET
   os=Ø
1270 PAUSE 20: PRINT AT cy*2,
   (cx-1)*9+5+os; BRIGHT 1; OVER
   1;"□"
1280 IF CODE INKEY$>31 AND CODE
   INKEY$<128 THEN LET
   s$(os+1)=INKEY$: PRINT AT cy*2,
   (cx-1)*9+5+os; s$(os+1): LET
   os=os+1: GOTO 1350
1290 IF CODE INKEY$=12 AND os>Ø THEN
   LET os=os-1: LET s$(os+1)=" ":
   PRINT AT cy*2,((cx-1)*9)+5+os; OVER
   Ø; BRIGHT Ø;"□□": GOTO 1270
1300 IF CODE INKEY$=13 AND
   s$="□□□□□□□□" THEN
   RETURN
1310 IF CODE INKEY$=13 AND t$="VAL"
   AND CODE s$(1)>64 THEN LET
   d$(wy+cy-1, wx+cx-1,17)="2":
   LET d$(wy+cy-1, wx+cx-1, TO
   8)=s$: LET y=cy-1: LET x=cx-1:
   GOSUB 1230: RETURN
1320 IF CODE INKEY$=13 AND t$="VAL"
   THEN GOSUB 1380: LET os=Ø: LET
   d$(wy+cy-1,wx+cx-1, TO 8)=s$:
   LET y=cy-1: LET x=cx-1: GOSUB
   1230: RETURN
1330 IF CODE INKEY$=13 AND t$="EQU"
   THEN GOSUB 1380: LET os=Ø: LET
   d$(wy+cy-1,wx+cx-1,9 TO 16)=s$:
   LET x=cx-1: LET y=cy-1: GOSUB
   1230: RETURN
1340 GOTO 1270
1350 IF os=8 AND t$="VAL" THEN BEEP
   .1,-10: GOSUB 1380: LET
   d$(wy+cy-1,wx+cx-1, TO 8)=s$:
   LET y=cy-1: LET x=cx-1: GOSUB
   1230: RETURN
1360 IF os=8 AND t$="EQU" THEN BEEP
   .1,-10: GOSUB 1380: LET
   d$(wy+cy-1,wx+cx-1,9 TO 16)=s$:
   LET y=cy-1: LET x=cx-1: GOSUB
   1230: RETURN
1370 GOTO 1270
1380 PRINT AT cy*2, ((cx-1)*9)+
   5+os; BRIGHT Ø; OVER 1;"□"
1390 IF t$="EQU" THEN GOSUB 1490:
   RETURN
1400 GOSUB 1410: PRINT AT cy*2,
   (cx-1)*9+5;s$: RETURN
1410 LET b$="□□□□□□□□
   ØØØ": FOR u=1 TO os: LET
   b$(u+(8-os))=s$(u): NEXT u
1420 FOR u=1 TO 11
1430 IF b$(u)<>"." THEN NEXT u
1440 IF u>=11 THEN LET b$(9)=".": LET
   s$=b$(4 TO ): RETURN
1450 LET w=6-u
1460 IF w<Ø THEN LET w=ABS w+1: LET
   s$=b$(w TO w+8): RETURN
1470 LET s$="□□□□□□□□"
1480 FOR u=1 TO 8-w: LET
   s$(w+u)=b$(u): NEXT u: RETURN
1490 IF s$(1)="#" THEN LET
   s$="□□□□□□□□": LET
   d$(wy+cy-1,wx+cx-1,17)="Ø":
   RETURN
1500 LET a$="": FOR z=1 TO 8: LET
   a$=a$+(s$(z) AND s$(z)<>"□")
1510 NEXT z: LET c=LEN a$
1520 IF c<4 THEN LET d$(wy+cy-1,
   wx+cx-1,17)="4": RETURN
1530 RESTORE 1630: FOR z=1 TO 11: LET
   f=Ø: READ m$
1540 FOR w=1 TO c
1550 IF m$(w)="A" THEN GOSUB 1650: IF
   f THEN GOTO 1610
1560 IF m$(w)="N" THEN GOSUB 1670: IF
   f THEN GOTO 1610
1570 IF m$(w)="O" THEN GOSUB 1690: IF
   f THEN GOTO 1610
1580 IF m$(w)="Z" THEN GOSUB 1710: IF f
   THEN GOTO 1610
1590 NEXT w: GOSUB 1140: IF NOT f THEN
   LET s$="□□□□□□□□": FOR
   w=1 TO c: LET s$(w)=a$(w): NEXT w:
   LET d$(wy+cy-1,
   wx+cx-1,18)=CHR$ z: LET
   d$(wy+cy-1,wx+cx-1,17)=
   "1": RETURN
1600 GOTO 1620
1610 NEXT z
1620 LET d$(wy+cy-1,wx+cx-1,
   17)="4": RETURN
1630 DATA "ANANOOO","ANNANOO",
   "ANANNOO","ANNANNO"
1640 DATA "ANZNOOOO","ANNZNOOO",
   "ANZNNOOO","ANNZNNOO",
   "ANZNNNOO","ANNZNNNO",
   "ANZNNNNO"
1650 IF a$(w)>="A" AND a$(w)<="X"
   THEN RETURN
1660 LET f=1: RETURN
1670 IF a$(w)>="Ø" AND a$(w)<="9"
   THEN RETURN
1680 LET f=1: RETURN
1690 LET c$=a$(w): IF c$="+" OR
   c$="-" OR c$="*" OR c$="/" OR
   c$="%" OR c$="$" OR c$="&"
   THEN RETURN
1700 LET f=1: RETURN
1710 IF a$(w)="Z" THEN RETURN
1720 LET f=1: RETURN
1730 FOR y=1 TO 30: FOR x=1 TO 24: LET
   d$(y,x, TO 8)="□□□□Ø.ØØ": LET
   d$(y,x,17)="Ø": LET d$(y,x,18)=CHR$
   Ø: NEXT x: NEXT y
1740 RESTORE 1750: FOR x=USR "a" TO
   USR "c"+7: READ a: POKE x,a: NEXT x:
   RETURN
1750 DATA 8,8,8,8,8,8,8,8
1760 DATA Ø,Ø,Ø,255,Ø,Ø,Ø,Ø
1770 DATA 8,8,8,255,8,8,8,8
1780 INPUT "FILENAME ?";n$
1790 SAVE n$ DATA d$()
1800 PRINT #1;AT Ø,Ø;"PRESS V TO
   VERIFY□"
1810 PAUSE Ø: IF INKEY$="v" OR
   INKEY$="V" THEN GOTO 1830
1820 RETURN
1830 VERIFY n$ DATA d$(): RETURN
1840 INPUT "FILENAME ?";n$
1850 LOAD n$ DATA d$(): RETURN
```

```
1720 S$=CHR$(13)+CHR$(145)+
   "□□□□□□□□□□□□□□□
   □□□□□□□███████
   ███████████████
   ███"
1730 W$="WRONG MODE FOR RELATIVE"
```

```
1740 GOSUB 590
1750 IF A$="←" THEN RETURN
1760 PRINT"ABSOLUTE OR RELATIVE";
1770 AA$="A":BB$="R":GOSUB 1280
1780 IF TP=0 AND A$="R" THEN PRINT
   S$;W$;:FOR W=1 TO 1000:NEXT
1790 IF TP=0 AND A$="R" THEN RETURN
1800 IF A$="A" THEN K1=0
1810 IF A$="R"THEN K1=1
1820 PRINT S$;
1830 PRINT"COLUMN OR ROW ALTER";
1840 AA$="C":BB$="R":GOSUB 1280
1850 IF A$="C" THEN K2=1
1860 IF A$="R" THEN K2=0
1870 PRINT S$;
1880 PRINT "▣FROM COL■";
1890 AA$="A":BB$="W":GOSUB 1250
1900 C1=ASC(A$)-64:PRINT A$
1910 INPUT"ROW";R1
1920 IF R1<0 OR R1>CM THEN PRINT
   TAB(6)"▣";:GOTO 1910
1930 PRINT"▣TO□□□";
1940 IF K2=0 THEN AA$="A":
   BB$="W":GOSUB 1250:C2=ASC
   (A$)-64:PRINTA$;:R2=R1
1950 IF K2=1THEN C2=C1:PRINT
   CHR$(C2+64):INPUT"□ROW□";
   R2
1960 IF R2<R1 OR R2>RM THEN 1950
1970 ON K1*2+K2+TP/2+1 GOSUB1990,
   2020,2070,2070,2080,2110,2140,2260
1980 RETURN
1990 GOSUB 2400
2000 FOR NN=C1 TO C2:GOSUB 2450:NEXT
   NN
2010 RETURN
2020 GOSUB 2400
2030 FOR NN=R1 TO R2
2040 A$=LEFT$(D$(NN,C1),8):
   IF ASC(A$)=128 THEN A$="□"+
   RIGHT$(A$,7)
2050 D$(NN,C1)=A$+F$
2060 NEXT NN
2070 RETURN
2080 GOSUB 2410
2090 FOR NN=C1 TO C2:GOSUB 2470:
   NEXT NN
2100 RETURN
2110 GOSUB 2410
2120 FOR NN=R1 TO R2:GOSUB 2440:
   NEXT NN
2130 RETURN
2140 GOSUB 2410:GOSUB 2420
2150 FF=0:IFX1>9 THEN FF=1
2160 AA$=MID$(F$,2,1+FF)
2170 BB$=RIGHT$(F$,5-FF)
2180 FOR NN=C1 TO C2
2190 F$=CHR$(64+Y1)+AA$+CHR$
   (64+Y2)+BB$
2200 IF LEN(F$)<8 THEN F$=F$+
   "□":GOTO 2200
2210 GOSUB 2470
2220 Y1=Y1+1:Y2=Y2+1:IF Y2<0 THEN
   Y2=-3
2230 IF Y2=24 THEN Y2=23
2240 NEXT NN
2250 RETURN
2260 GOSUB 2410:GOSUB 2420
2270 FF=0:IF X1>9 THEN FF=1
2280 IF X2>9 THEN FF=FF+1
2290 AA$=MID$(F$,5+FF,2)
2300 FOR NN=R1 TO R2
2310 X1$=RIGHT$(STR$(X1),LEN
   (STR$(X1))-1)
2320 X2$=RIGHT$(STR$(X2),LEN
   (STR$(X2))-1)
2330F$=CHR$(Y1+64)+X1$+CHR$
   (Y2+64)+X2$+AA$
2340 IF LEN(F$)<8 THEN F$=F$+
   "□":GOTO 2340
2350 GOSUB 2440
2360 X1=X1+1:X2=X2+1:IF Y2=-3
   THEN X2=X2-1
2370 IF Y2=24 THEN X2=X2-1
2380 NEXT NN
2390 RETURN
2400 F$=RIGHT$(D$(R,C),8):
   RETURN
2410 F$=LEFT$(D$(R,C),8):
   RETURN
2420 GOSUB 2480:IF X1=0 THEN GOSUB
   2080:RETURN
2430 RETURN
2440 D$(NN,C1)=F$+RIGHT$(D$(NN,C1),
   8):RETURN
2450 A$=LEFT$(D$(R1,NN),8):IF
   ASC(A$)=128 THEN A$="□"+
   RIGHT$(A$,7)
2460 D$(R1,NN)=A$+F$:RETURN
2470 D$(R1,NN)=F$+RIGHT$(D$(R1,
   NN),8):RETURN
2480 F=0
2490 Y1=ASC(MID$(F$,1,1))-64
2500 X1=VAL(MID$(F$,2,2))
2510 IF X1>9 THEN F=1
2520 Y2=ASC(MID$(F$,(3+F),
   1))-64
2530 X2=VAL(MID$(F$,(4+F),2))
2540 IF X2>9 THEN F=F+1
2550 Q1$=OP$:Q2$=MID$(F$,
   5+F,2)
2560 GOSUB 2670
2570 D=VAL(MID$(F$,6+F,1))
2580 RETURN
2590 PRINT"TO START FROM:
   COL+ROW■";
2600 AA$="A":BB$="W":GOSUB 1250
2610 PRINT A$:CS=ASC(A$)-64
2620 INPUT A$:V=VAL(A$)
2630 IF V>RM-LM THEN V=RM-LM+1
2640 IF V<1 THEN V=1
2650 RS=V
2660 RETURN
```



Entering equations on the Spectrum



Calculating the result on the values screen

```
2670 Q=Ø
2680 Q=Q+1
2690 IF MID$(Q1$,Q,1)=Q2$ THEN RETURN
2700 IF Q<LEN(Q1$) THEN 2680
2710 Q=Ø:RETURN
```

```
1770 REM arithmetic routines
1780 ans=FNv1+FNv2:RETURN
1790 ans=FNv1-FNv2:RETURN
1800 ans=FNv1*FNv2:RETURN
1810 IF FNv2=Ø THEN ans=Ø:
   RETURN ELSE ans=FNv1/FNv2:
   RETURN
1820 ans=FNv2*FNv1/100:RETURN
1830 ans=Ø:FOR n%=r1% TO r2%
1840 ans=ans+VAL(MID$(D$(n%,c1%),9,
   8))
1850 NEXT
1860 RETURN
1870 ans=Ø:FOR n%=c1% TO c2%
1880 ans=ans+VAL(MID$(D$(r1%,n%),9,8))
1890 NEXT
1900 RETURN
1910 ans =FNv1:RETURN
1920 DEF PROCsave
1930 CLS
1940 PRINTTAB(Ø,5)"Do you wish to
   save"'"current data (Y/N)?□";
1950 PROCgetabs ("Y", "N")
1960 IF A$="N" ENDPROC
1970 X=OPENOUT("SprdDta")
1980 PRINT#X,Ø,Ø,"***"
1990 FOR c=1 TO Cols
2000 FOR r=1 TO Rows
2010 IF LEFT$(D$(r,c),1)<>CHR$128
   PRINT#X,r,c,D$(r,c)
2020 NEXT,
2030 CLOSE#X
2040 ENDPROC
2050 DEF PROCload
2060 PRINTTAB(Ø,5)"Do you wish to load"'"
   an existing file (Y/N)?□";
2070 PROCgetabs ("Y","N")
2080 IF A$="N" ENDPROC
2085 PRINT'"PRESS PLAY ON RECORDER"
2090 X=OPENIN("SprdDta")
2100 REPEAT
2110 INPUT#X,r,c,D$(r,c)
2120 UNTIL EOF#X
2130 CLOSE#X
2140 ENDPROC
2150 DEF PROCclrp(A$,vpos)
2160 PRINTTAB(Ø,vpos)SPC2Ø
   TAB(Ø,vpos)A$;
2170 ENDPROC
2180 DEF PROCreplicate
2190 vpos=VPOS
2200 PROCcellin(vpos)
2210 PRINTTAB(Ø,vpos+1)CHR$(129);CHR$
   (Col+64);Row;
2220 PROCclrp ("Absolute or
   Relative",vpos):PROCgetabs
   ("A","R")
2230 IF Type=Ø AND A$="R" PRINT'
   CHR$129;"Wrong mode for relative":
   A$=INKEY$(200):ENDPROC
2240 IF A$="A" k1%=Ø:A$="Abs"
2250 IF A$="R" k1%=1:A$="Rel"
2260 PRINTTAB(7,vpos+1)A$
2270 PROCclrp("Column or Row alter□",
   vpos):PROCgetabs("C","R")
2280 IF A$="C" k2%=1:A$="Col"
2290 IF A$="R" k2%=Ø:A$="Row"
2300 PRINTTAB(12,vpos+1)A$
2310 PROCclrp("From□",vpos)
2320 PROCgetbet("@","Y"):c1%=
   ASC(A$)-64:PRINTA$;
2330 REPEAT PRINTTAB(6,vpos)SPC(4);:
   VDU8,8,8,8:INPUT""r1%:UNTIL r1%>Ø
   AND r1%<=Rows
2340 PRINTTAB(18,vpos+1)CHR$(c1%+
   64);r1%
2350 PROCclrp("To□□□",vpos)
2360 IF k2%=1 PRINTCHR$(c1%+64);:
   REPEAT PRINTTAB(6,vpos)SPC(4);:VDU8,8,
   8,8:INPUT""r2%:UNTIL r2%>r1% AND
   r2%<=Rows:c2%=c1%
2370 IF k2%=Ø PROCgetbet("@","Y"):
   c2%=ASC(A$)-64:PRINTA$;:r2%=r1%
2380 PRINTTAB(22,vpos+1)CHR$(c2%+
   64);r2%
2390 ON k1%*2+k2%+Type/2+1 GOSUB
   241Ø,244Ø,249Ø,249Ø,250Ø,253Ø,256Ø,
   268Ø
2400 ENDPROC
2410 GOSUB2800
2420 FOR n%=c1% TO c2%:GOSUB
   2850:NEXT
2430 RETURN
2440 GOSUB 2800
2450 FOR n%=r1% TO r2%
2460 A$=LEFT$(D$(n%,c1%),8):IF ASC
   (A$)=128 A$="□"+RIGHT$(A$,7)
2470 D$(n%,c1%)=A$+F$
2480 NEXT
2490 RETURN
2500 GOSUB2810
2510 FOR n%=c1% TO c2%:GOSUB
   2870:NEXT
2520 RETURN
2530 GOSUB2810
2540 FOR n%=r1% TO r2%:GOSUB
   2840:NEXT
2550 RETURN
2560 GOSUB2810:GOSUB2820
2570 f%=Ø:IF x1%>9 f%=1
2580 a$=MID$(F$,2,1+f%)
2590 b$=RIGHT$(F$,5-f%)
2600 FOR n%=c1% TO c2%
2610 F$=CHR$(64+y1%)+a$+CHR$
   (64+y2%)+b$
2620 IF LENF$<8 F$=F$+"□":
```



The equations screen on the Acorn



Entering values and labels

```
   GOTO262Ø
2630 GOSUB 287Ø
2640 y1%=y1%+1:y2%=y2%+1:
   IF y2%<Ø y2%=-3
2645 IF y1%=24 OR y2%=24 THEN
   n%=c2%: GOTO 266Ø
2650 IF y2%=25 y2%=24
2660 NEXT
2655 IF y1%=25 y1%=24
2670 RETURN
2680 GOSUB281Ø:GOSUB282Ø
2690 f%=Ø:IF x1%>9 f%=1
2700 IF x2%>9 f%=f%+1
2710 a$=MID$(F$,5+f%,2)
2720 FOR n%=r1% TO r2%
2730 F$=CHR$(y1%+64)+STR$(x1%)+
   CHR$(y2%+64)+STR$(x2%)+a$
2740 IF LENF$<8 F$=F$+"□":
   GOTO 274Ø
2750 GOSUB 284Ø
2760 x1%=x1%+1:x2%=x2%+1:IF
   y2%=-3 x2%=x2%-1
2765 IF x1%=21 OR x2%=21 THEN n%=
   r2%:GOTO 278Ø
2770 IF y2%=24 x2%=x2%-1
2775 IF y1%=24 x1%=x1%-1
2780 NEXT
2790 RETURN
2800 F$=RIGHT$(D$(Row,Col),8):RETURN
2810 F$=LEFT$(D$(Row,Col),8):RETURN
2820 PROCdecode:IF x1%=Ø GOSUB25ØØ:
   RETURN
2830 RETURN
2840 D$(n%,c1%)=F$+RIGHT$(D$(n%,
   c1%),8):RETURN
2850 A$=LEFT$(D$(r1%,n%),8):IF
   ASC(A$)=128 A$="□"+RIGHT$
   (A$,7)
2860 D$(r1%,n%)=A$+F$:RETURN
2870D$(r1%,n%)=F$+RIGHT$(D$(r1%,
   n%),8):RETURN
2880 DEF PROCdecode
2890 f%=Ø
2900 y1%=FNc(1):x1%=FNr(2)
2910 IF x1%>9 f%=1
2920 y2%=FNc(3+f%):x2%=FNr(4+f%)
2930 IF x2%>9 f%=f%+1
2940 op%=INSTR(Op$,MID$(F$,5+f%))
2950 d%=FNr(6+f%)
2960 ENDPROC
2970 DEF PROCwindowstart
2980 PRINT"To start from:
   Col+Row□";:PROCgetbet("@","Y")
2990 PRINTA$;:Colstart=ASC(A$)-64
3000 IFColstart>Cols-3 THEN
   Colstart=Cols-3
3010 INPUT""A$:v=VALA$
3020 IF v>Rows-Length□v=Rows-
   Length+1
3030 IF v<1 v=1
3040 Rowstart=v
3050 ENDPROC
3060 IF ERR=17 THEN 17Ø
3070 REPORT:PRINT"□AT LINE□";
   ERL:*FX4
3080 END
```

CHANGES FOR THE ELECTRON

You'll need to make a few changes to the Acorn program so that it will run on the Electron. Delete Lines 14Ø, 36Ø, 44Ø to 48Ø and 77Ø then change:

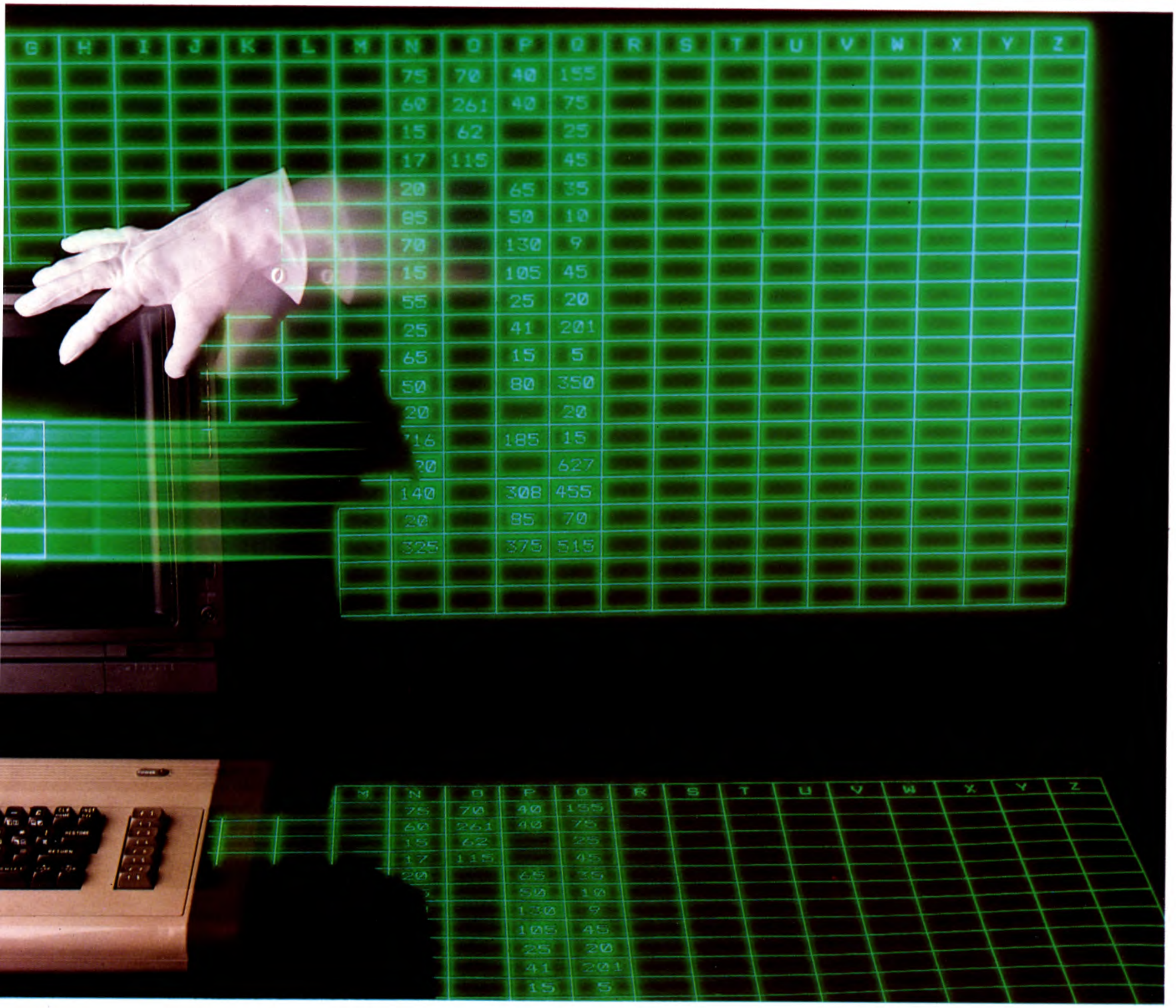
```
10 MODE 6:*FX4,1
50 DIM D$(Rows,Cols):VDU 23,128,Ø,Ø,Ø,Ø,
   Ø,Ø,Ø,Ø
370 IF Type=Ø PRINT"□"RIGHT$(D$
   (r,c),8);
380 IF Type=8 PRINT"□"LEFT$(D$
   (r,c),8);
540 PRINT"<TAB> to exit□□:□";
1310 PRINT"WORKING"
2210 PRINTTAB(Ø,vpos+1)CHR$(Col+64);
   Row;
2230 IF Type=Ø AND A$="R"
   PRINT'"Wrong mode for relative":
   A$=INKEY$(2ØØ):ENDPROC
```

T

```
1490 PRINT @448: PRINT @448,;: IF LEN
   (D$(CC, CR))=1 THEN PRINT
   "NOTHING TO COPY": SOUND5,5: FOR
   K=1 TO 5ØØ: NEXT: RETURN ELSEPRINT
   "aBSOLUTE OR rELATIVE COPY";
1500 A$=INKEY$:IF A$="" THEN15ØØ
1510 IF A$=CHR$(13) THEN RETURN
1520 IF A$="A" THEN TC=1:
   GOTO154Ø
1530 IF A$<>"R" THEN 149Ø ELSE
   TC=Ø
1540 PRINT@448
1550 IF TC=Ø AND ASC(D$(CC,CR))
   <>131 THEN PRINT@448,"WRONG
   COPY MODE — EQUATIONS ONLY";:
   SOUND 1,5:FORD=1TO5ØØ:NEXTD:GOTO
   49Ø
1560 PRINT@448:PRINT@448,"cOLUMN
   OR rOW COPY";
1570 A$=INKEY$:IF A$="" THEN157Ø
1580 IF A$=CHR$(13) THENRETURN
1590 IF A$="C" THEN DC=1:GOTO 161Ø
1600 IF A$<>"R" THEN 156Ø ELSE
   DC=Ø
1610 PRINT@448:PRINT@448,"START AT
   CELL";:INPUT A$:IF A$="" THEN
   RETURN
1620 S1=ASC(A$)-64:IF S1<1 OR
   S1>25 THEN 161Ø
1630 S2=VAL(MID$(A$,2)):IF S2<1 OR
   S2>3Ø THEN 161Ø
1640 PRINT@448:PRINT@448,"FINISH AT
   CELL";:INPUT A$
```




```
1650 IF A$="" THENRETURN
1660 C1=ASC(A$)-64:IF C1<1 OR
   C1>25 THEN 164Ø
1670 C2=VAL(MID$(A$,2)):IF C2<1 OR
   C2>3Ø THEN 164Ø
1680 PRINT@448,"COPYING — WAIT"
1690 IF TC=1 THEN 197Ø
1700 IF DC=1 THEN 184Ø
1710 IF C2<>S2 THEN 2Ø9Ø
1720 AS=ASC(MID$(D$(CC,CR),
   2))-64:IF AS<>26 AND
   AS+C1-S1>25 THEN
   C1=25+S1-AS
1730 AS=ASC(MID$(D$(CC,CR),
   5))-64:IF AS<>26 AND
   AS+C1-S1>25 THEN
   C1=25+S1-AS
```




```
1740 D$(S1,S2)=D$(CC,CR)
1750 FOR I=S1+1 TO C1
1760 A$=D$(I-1,S2)
1770 IF MID$(A$,2,1)="Z" THEN 1790
1780 MID$(A$,2,1)=CHR$(ASC(MID$
   (A$,2,1))+1)
1790 IF MID$(A$,5,1)="Z" THEN 1810
1800 MID$(A$,5,1)=CHR$(ASC(MID$
   (A$,5,1))+1)
1810 D$(I,S2)=A$:NEXT
1820 IF C1>CX THEN CX=C1
1830 GOTO 2080
1840 IF C1<>S1 THEN 2090
1850 AS=VAL(MID$(D$(CC,CR),3,2)):IF
   AS+C2-S2>30 THEN
   C2=30+S2-AS
1860 AS=VAL(MID$(D$(CC,CR),6)):IF
   AS+C2-S2>30 THEN
   C2=30+S2-AS
1870 D$(S1,S2)=D$(CC,CR)
1880 FOR I=S2+1 TO C2
1890 A$=D$(S1,I-1)
1900 IF MID$(A$,2,1)="Z" THEN 1920
1910 V=VAL(MID$(A$,3,2))+1:IF V<10
   THEN MID$(A$,3,2)=STR$(V) ELSE
   MID$(A$,3,2)=MID$(STR$(V),2,2)
1920 IF MID$(A$,5,1)="Z" THEN 1940
1930 V=VAL(MID$(A$,6,2))+1:IF V<10
   THEN MID$(A$,6,2)=STR$(V) ELSE
   MID$(A$,6,2)=MID$(STR$(V),2,2)
1940 D$(S1,I)=A$:NEXT
1950 IF C2>RX THEN RX=C2
1960 GOTO 2080
1970 IF DC=1 THEN 2030
1980 IF C2<>CR THEN 2090
1990 FOR I=CC+1 TO C1
2000 D$(I,CR)=D$(I-1,CR):
   NEXT
2010 IF C1>CX THEN CX=C1
2020 GOTO 2080
2030 IF C1<>CC THEN 2090
2040 FOR I=CR+1 TO C2
2050 D$(CC,I)=D$(CC,I-1):
   NEXT
2060 IF C2>RX THEN RX=C2
2070 GOTO 2080
2080 MO=1:GOSUB70:RETURN
2090 PRINT@448,"error IN
   DESTINATION":SOUND1,5:FORD=1
   TO500:NEXTD
2100 GOTO 2080
```

SOLIDS OF ROTATION

Here is a handy technique that lets you turn a simple outline into a three-dimensional shape. Use it to design anything from a set of cocktail glasses to a top hat



Drawing symmetrical objects, especially in three dimensions, is normally very tricky. But a computer can make light work of the task. With this program all you have to do is draw the outline of one side of the shape, and the computer draws the rest for you, filling in the shape to create a wire-frame solid-looking object.

The program works by rotating your original outline about a central axis. So you can create anything with a circular cross-section—such as a vase or bowl, a glass, a candlestick or bell, a top hat, an apple or orange, or a thousand other things. Because the program *rotates* your original line, the solid shape produced is called a *solid of rotation*. The program also lets you view the object from any angle, and as an extra bonus (except with the Commodore) it introduces some animation into the picture by spinning the object about its axis.

DRAWING THE SHAPE

Drawing the outline of your shape is very easy—the program uses the *rubber-banding* technique, where you can stretch and move a line about until you're happy with its position. Rubber-banding allows interactive control over the shape you're drawing, letting you see by eye when each line is in the right place. (See pages 998 to 1003 for more information on rubber-banding.)

You can draw up to 20 lines in the outline. This is usually more than enough and most shapes can be drawn with about half a dozen lines. However, curves in the outline have to be made up from a series of short straight lines and this will take up more lines than a simple angular shape.

A POINT OF VIEW

The program also gives you control over the direction from which you look at the object—not only from above, below, or straight on, but from anywhere else as you can specify the exact angle of view (an angle of about 70° gives a good view from above, as though you



were looking at the object on a table). You can change the angle at any time and the program redraws the solid of rotation with the new orientation.

The program stores the coordinates of all your lines. When you have finished drawing it takes each line in turn and rotates it about the centre point in steps of 18 degrees—giving 20 steps around the whole circle. The angle of view is also taken into account and the circle appears more and more flattened as the viewpoint gets lower. The finished effect can be seen in the screen pictures and drawings on the following pages.

Once the first view has been drawn the program waits. If you press the space bar or the space key then you can choose to view the same shape from a different angle.

ADDING SOME SPIN

On all but the Commodore you can also see the object spin by pressing one of the other keys. To make the shape spin, the program first saves the picture in memory using paged graphics techniques (see pages 1024 to 1029) and 1132 to 1137). The solid is then redrawn slightly differently as though it had been rotated a few degrees. This page is then saved as well and the procedure is repeated several times. Once all the pages have been drawn and saved in memory they are called back and displayed one after the other, making the object appear to rotate.

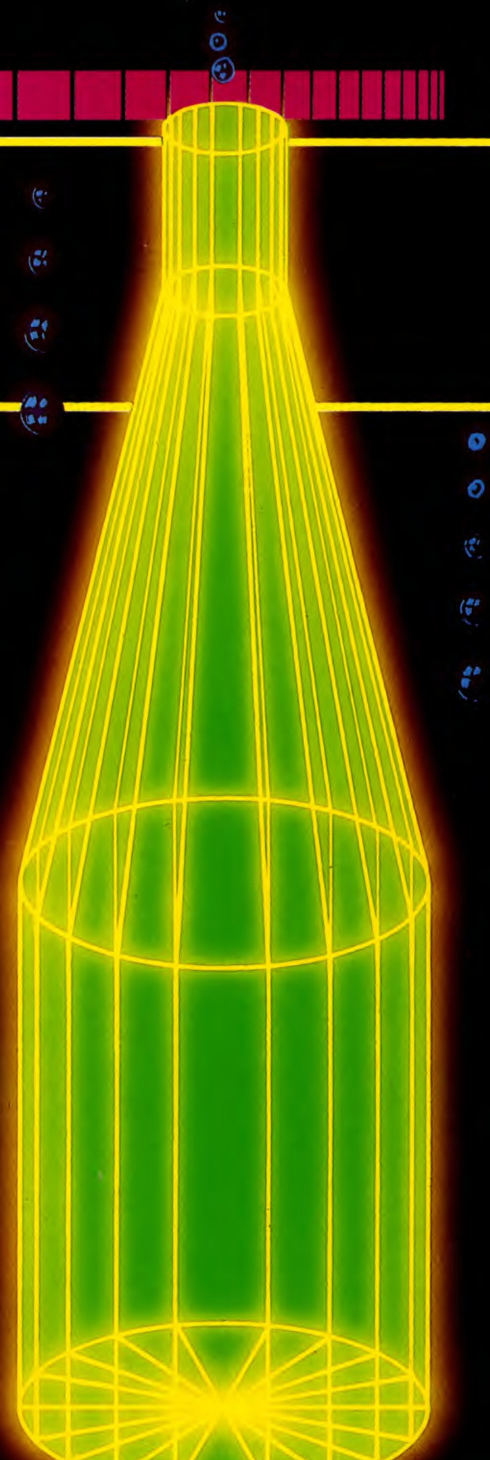
The reason the Commodore program doesn't do this is because, when using paged graphics, the amount of screen that can be used is rather small and, given the screen's limited resolution, the lines making up the object would tend to blur together too much to be visible. By dispensing with the spinning effect the object can be made much larger and so clearer.

USING THE PROGRAM

Enter the program now and try drawing some shapes. The procedure for drawing is to move the cursor to the place you want a line to start at, then press an appropriate key to mark the starting point. Then move the cursor to



- THREE-DIMENSIONAL DRAWINGS
- CREATING SOME SHAPES
- DRAWING THE OUTLINE
- RUBBER-BANDING
- ALTERING THE VIEWPOINT
- SPINNING THE SHAPE
- PAGED GRAPHICS
- ENTERING THE PROGRAM
- HOW IT WORKS



manipulate the 'rubber-band' line on the screen. When you're happy with its position, and want to 'fix' it in place you press another key. Continue like this—moving and stretching the rubber-band and fixing lines—to build up the profile.

On the Spectrum use the cursor keys to move the cursor; M to mark the starting position and ENTER to 'fix' each section of the profile. Press Q when the design is complete to move on to the next stage, the running of the program.

On the Commodore use the Z X ; and / keys to move the cursor. Use the ← symbol to mark the start position and SPACE to fix each line. Press RETURN when you've finished.

On the Acorn, Z X P and L move the cursor and SPACE fixes the line. The first time you press SPACE no line is drawn so use this to mark the start position. Press RETURN to end.

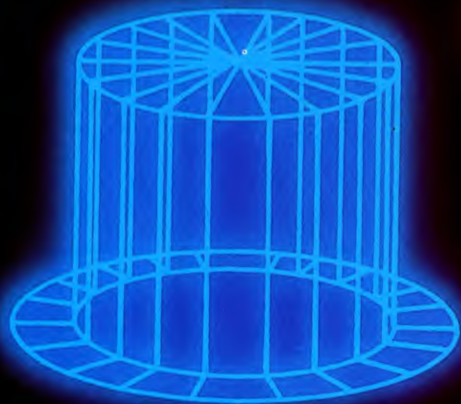
Finally, on the Dragon and Tandy the arrow keys move the cursor, ENTER marks the start position, SPACE fixes a line and Y ends.

When you draw the shape you can speed up movement of the cursor by simultaneously pressing CAPS SHIFT on the Spectrum, SHIFT on the Commodore and Acorn, and CLEAR on the Dragon and Tandy.

```
10 PRINT AT 0,10; INK 2; PAPER 5;"3D
   ROTATION": PAUSE 100
20 PAUSE 2500
30 CLEAR 30000: PAPER 0: INK 5: CLS :
   POKE 23658,0
40 GOSUB 2000: BORDER 0
50 PRINT AT 0,0; PAPER 6; INK
   0;"□□□□□MOVE LINE = CURSOR
   KEYS□□□□□□□(CAPSHIFT FOR
   DOUBLE SPEED MOVE)"
60 PRINT : PRINT PAPER 6; INK
   0;"□□□□□FIX LINE = ENTER
   KEY□□□□□□□□""□FIX
   CURSOR = M□□□FINISH = Q□□
   □□"
70 INPUT "ENTER ANGLE OF VIEW
   (0-180)";i: IF i<0 OR i>180 THEN
   GOTO 70
80 PLOT 60,100: DRAW 135,0: DRAW
   0,-49: DRAW -135,0: DRAW 0,49
90 LET is = SIN (i/180*PI)
100 LET bf = 0: LET a$ = ""
110 LET x = 128: LET y = 51
120 LET xx = x: LET yy = y
130 OVER 1
140 PLOT x,y: DRAW xx-x,yy-y
150 PLOT x,y: DRAW xx-x,yy-y
160 LET z$ = INKEY$: IF z$ = "" THEN GOTO
   140
170 LET z = CODE z$
180 IF z = 13 THEN GOSUB 500
190 IF z = 113 THEN GOTO 600
195 IF z = 109 AND bf <> 1 THEN PLOT
   255-x,y: LET x = xx: LET y = yy: LET
   bf = 1
200 IF z = 53 AND xx > 128 THEN LET
   xx = xx-1
210 IF z = 8 AND xx > 130 THEN LET
   xx = xx-2
220 IF z = 55 AND yy < 100 THEN LET
   yy = yy+1
230 IF z = 11 AND yy = 98 THEN LET
   yy = yy+2
240 IF z = 56 AND xx < 195 THEN LET
   xx = xx+1
250 IF z = 9 AND xx < 194 THEN LET
   xx = xx+2
260 IF z = 54 AND yy > 52 THEN LET
   yy = yy-1
270 IF z = 10 AND yy > 51 THEN LET
   yy = yy-2
280 GOTO 140
500 OVER 0: PLOT x,y: DRAW xx-x,
   yy-y
510 PLOT 255-x,y: DRAW x-xx,
   yy-y
520 LET x = xx: LET y = yy: LET
   a$ = a$ + CHR$ (x-128) + CHR$
   (y-51): OVER 1: RETURN
600 INK 7: CLS : OVER 0: DIM a(20,2)
610 FOR a = 0 TO 7
620 PRINT AT 21,3; PAPER 2; BRIGHT
   1;"DRAWING PICTURE□";a+1;"□OUT
   OF 8"
630 GOSUB 1000
640 IF a <> 0 THEN GOTO 670
650 PRINT AT 20,0; INVERSE 1;"□SPACE
   KEY TO ALTER VIEW
   ANGLE□""□□□ANY OTHER KEY TO
   CONTINUE□□□"
660 LET X$ = INKEY$: IF X$ = "" THEN
   GOTO 650
662 IF X$ <> "□" THEN GOTO 670
664 INPUT "ENTER NEW ANGLE
   (0-180)";i
666 IF i < 0 OR i > 180 THEN GOTO 662
668 CLS : LET is = SIN (i/180*PI):
   GOTO 620
670 GOSUB 1600
680 CLS
690 NEXT a
700 PRINT AT 20,11; PAPER 7; INK
   1;"□FRAME :□";
```







```
710 FOR a=128 TO 240 STEP 16
720 POKE 30114,a: RANDOMIZE USR 30112
730 PRINT AT 20,19;a/16
740 NEXT a: GOTO 710
1000 FOR b=1 TO LEN a$ STEP 2
1010 LET x=CODE a$(b)
1020 LET y=CODE a$(b+1)
1030 GOSUB 1500
1040 NEXT b
1050 RETURN
1500 FOR c=0 TO 399 STEP 20
1510 LET d=c+(2.25*a):LET py=is*y
1515 IF d>=360 THEN LET d=d-360
1520 GOSUB 1530: NEXT c
1525 DRAW bx-xd,by-yd: RETURN
1530 LET yd=SIN (d/180*PI)*x*COS
   (i/180*PI)
1532 LET xd=COS (d/180*PI)*x
1535 LET xd=128+xd: LET
   yd=90+yd+py
1540 IF c=0 THEN PLOT xd,yd: LET bx=xd:
   LET by=yd
1550 DRAW xd-PEEK 23677,yd-PEEK
   23678
1560 IF b=1 AND bf=1 THEN GOTO 1580
1565 IF b=1 THEN PLOT 128,90: DRAW
   xd-128,yd-90: GOTO 1580
1570 PLOT a(1+c/20,1),a(1+c/20,2): DRAW
   xd-a(1+c/20,1),yd-a(1+c/20,2)
1580 LET a(1+c/20,1)=xd
1585 LET a(1+c/20,2)=yd
1590 RETURN
1600 POKE 30102,128+a*16
1610 RANDOMIZE USR 30100
1620 RETURN
2000 RESTORE 2050
2010 FOR a=0 TO 23
2020 READ b: POKE a+30100,b
2030 NEXT a
2040 RETURN
2050 DATA 17,0,0,33,0,64,1,0,16,237,176,
   201
2060 DATA 33,0,0,17,0,64,1,0,16,237,176,
   201
```

You will need a Simons' Basic cartridge or *INPUT*'s hi-res graphics utility in order to RUN this program.

```
10 HIRES 3,6:COLOUR 6,6:DIM A(19,1):
   POKE 650,128
20 RD=ATN(1)/45
40 BX=141:BY=190:XX=BX:YY=BY:
   X=BX:Y=BY
50 REC 59,141,163,51,2
60 LINE BX,BY,X,Y,2
70 GET K$:CP=PEEK(197):IF K$="□"
   THEN GOSUB 500
80 M=2:IF PEEK(653)=1 THEN M=8
90 LINE BX,BY,X,Y,2:IF K$="←" AND
   A$="" THEN BX=X:BY=Y:BF=1:
   GOSUB 500
100 IF K$=CHR$(13) AND LEN(A$)>0
   THEN 160
110 IF CP=50 AND Y-M>142 THEN
   Y=Y-M
120 IF CP=55 AND Y+M<191 THEN
   Y=Y+M
130 IF CP=12 AND X-M>140 THEN
   X=X-M
140 IF CP=23 AND X+M<220 THEN
   X=X+M
150 GOTO 60
160 GOSUB 2000:HIRES 0,0:MULTI 3,6,14:
   COLOUR 6,0
170 GOSUB 1000:COLOUR 14,0
190 GET K$:IF K$="□" THEN 160
195 GOTO 190
500 LINE BX,BY,X,Y,1
510 LINE 281-BX,BY,281-X,Y,1
520 BX=X:BY=Y:A$=A$+CHR$
   (X-141)+CHR$(190-Y):RETURN
1000 FOR B=1 TO LEN(A$) STEP 2
1010 X=ASC(MID$(A$,B,1)):Y=ASC(MID$
   (A$,B+1,1))
1020 GOSUB 1500:NEXT B
1040 RETURN
1500 FOR C=0 TO 399 STEP 20
1510 D=C:PY=IS*(Y*.5)
1520 GOSUB 1530:NEXT C:RETURN
1530 YD=100-SIN(D*RD)*X*COS(I*RD)-
   PY:XD=80+COS(D*RD)*X
1540 IF C=0 THEN BX=XD:BY=YD:
   XX=XD:YY=YD
1550 LINE XX,YY,XD,YD,1:XX=XD:YY=YD
1560 IF B=1 AND BF=1 THEN 1580
1565 IF B=1 THEN LINE 80,91,XD,YD,2:
   GOTO 1580
1570 LINE A(C/20,0),A(C/20,1),XD,YD,3
1580 XX=XD:YY=YD:A(C/20,0)=XD:
   A(C/20,1)=YD:RETIRN
2000 NRM:I=0:INPUT "◣◘ANGLE OF
   VIEW (0-180)";I
2010 IF I<0 OR I>180 THEN 2000
2020 IS=SIN(I*RD):RETURN
```

Note the changes below for the Electron.

```
10 DIM A(19,1),X(20),Y(20)
20 MODE1:VDU23;8202;0;0;0;
40 BX=644:BY=104:X=BX:Y=BY:N=0
50 PROCDRAW
60 GOSUB2000:MODE4:
   VDU23;8202;0;0;0;
70 HIMEM=&3000:?&34E=&30
80 VDU23;6,16,0;0;0;
90 A=0:PROCSCREEN
100 VDU 24,0;512;1279;1023;:FOR A=3 TO
   0 STEP-1:CLG:GOSUB1000:IF A=3
   THEN G=GET:IF G=32 THEN A=0:
   NEXT:MODE4:GOTO 60
110 PROCSCREEN:NEXT
```

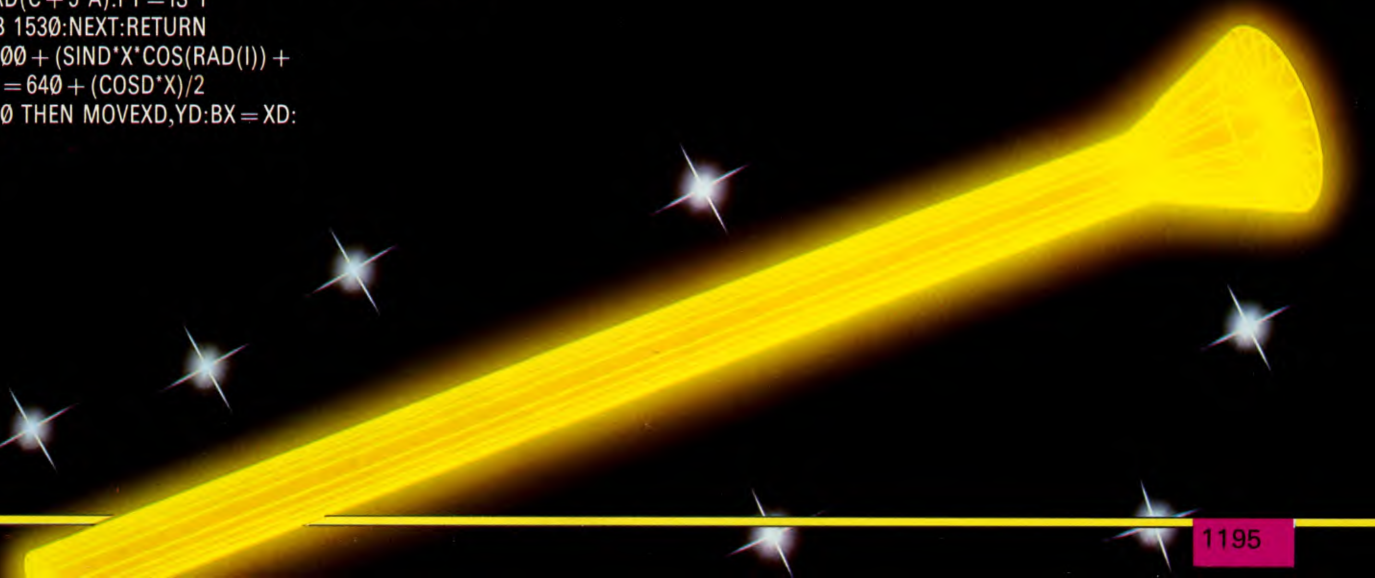
```
120 FOR A=Ø TO 3:D=INKEY(1Ø):
   PROCSCREEN:NEXT
130 IF INKEY(1)=32 THEN GOSUB 2ØØØ:
   GOTO 9Ø ELSE 12Ø
140 DEF PROCSCREEN
145 *FX19
150 X%=&6C-A*&14:?&351=X%:
   ?&35Ø=Ø:X%=X%*32
160 VDU 23;13,X%MOD256,Ø;Ø;Ø;23;12,X%
   DIV256,Ø;Ø;Ø;
180 ENDPROC
500 DEF PROCDRAW
510 MOVE8Ø,1ØØ:DRAW8Ø,5ØØ:DRAW 12ØØ,
   5ØØ:DRAW 12ØØ,1ØØ:DRAW8Ø,1ØØ
520 GCOL3,3:IF N<>Ø THEN MOVE
   BX,BY:DRAWX,Y□ELSE PLOT69,X,Y
530 IF INKEY(-99) THEN PROCIN
540 IF INKEY(-1) THEN M=16 ELSE M=4
550 IF N<>Ø THEN MOVE BX,BY:DRAW
   X,Y□ELSE PLOT69,X,Y
560 IF INKEY(-74) AND N>Ø OR
   N=2Ø THEN ENDPROC
570 IF INKEY(-87) AND Y-M>1ØØ THEN
   Y=Y-M
580 IF INKEY(-56) AND Y+M<5ØØ THEN
   Y=Y+M
590 IF INKEY(-98) AND X-M>636 THEN
   X=X-M
600 IF INKEY(-67) AND X+M<12ØØ
   THEN X=X+M
610 GOTO 52Ø
620 DEF PROCIN
630 IF N=Ø THEN 66Ø
640 GCOLØ,3:MOVEBX,BY:DRAWX,Y
650 MOVE128Ø-BX,BY:DRAW 128Ø-X,Y
660 BX=X:BY=Y:X(N)=X-64Ø:Y(N)=
   Y-1ØØ:N=N+1:REPEATUNTIL
   INKEY(-99)=Ø:ENDPROC
1000 FOR B=Ø TO N-1
1010 X=X(B):Y=Y(B)
1020 GOSUB15ØØ:NEXT
1040 RETURN
1500 FOR C=Ø TO 36Ø STEP 2Ø
1510 D=RAD(C+5*A):PY=IS*Y
1520 GOSUB 153Ø:NEXT:RETURN
1530 YD=7ØØ+(SIND*X*COS(RAD(I))+
   PY)/2:XD=64Ø+(COSD*X)/2
1540 IF C=Ø THEN MOVEXD,YD:BX=XD:
   BY=YD
1550 DRAW XD,YD
1560 IF B=Ø THEN MOVEXD,YD:GOTO 158Ø
1570 MOVE A(C/2Ø,Ø),A(C/2Ø,1):DRAWXD,
   YD
1580 A(C/2Ø,Ø)=XD:A(C/2Ø,1)=YD:
   RETURN
2000 CLS:*FX15,Ø
2010 INPUT"ANGLE OF VIEW (Ø-18Ø)□",I
2020 IFI<ØORI>18ØTHEN2ØØØ
2030 IS=SIN(RAD(I)):RETURN
```

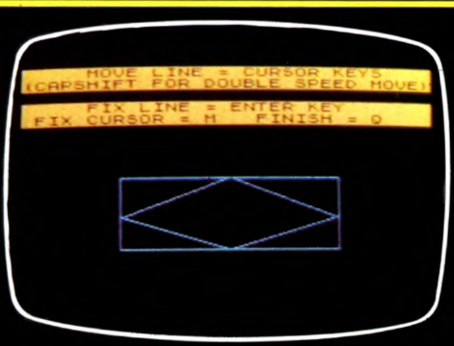
On the Electron, if you want to see the object spin, delete Lines 8Ø and 145 and then enter the following lines:

```
30 PROCASS
90 VDU 23;13,128,Ø;Ø;Ø;23;12,13,Ø;Ø;Ø;:
   ?&351=&6C:?&35Ø=Ø
120 FOR A=1 TO 3:PROCSCREEN:NEXT
150 X%=&6C-A*&14
160 Y%=&6C
170 CALL SWITCH
180 ENDPROC
200 DEF PROCASS
210 DIM SWITCH 1ØØ
220 FOR T=Ø TO 2 STEP 2
230 P%=SWITCH
240 [OPT T
250 STX &71
260 STY &73
270 LDX #2Ø
280 LDY #Ø
290 STY &7Ø
300 STY &72
310 .L2□LDA (&7Ø),Y
320 PHA
330 LDA (&72),Y
340 STA (&7Ø),Y
350 PLA
360 STA (&72),Y
370 INY
380 BNE L2
390 INC &71
400 INC &73
410 DEX
420 BNE L2
430 RTS
440 ]NEXT
450 ENDPROC
```

For the Tandy, change the 223 in Line 2Ø to 247.

```
10 PMODE2,1:PCLEAR2:CLEAR2ØØ,7679:
   DIMA(19,1)
20 RD=ATN(1)/45:V=223
30 PCLS1:SCREEN1,Ø
40 BX=128:BY=19Ø:DRAW"BM128,19Ø":
   X=BX:Y=BY
50 LINE(31,191)-(255,14Ø),PRESET,B
60 LINE(BX,BY)-(X,Y),PRESET
70 IFPEEK(345)=V GOSUB5ØØ
80 IFPEEK(339)=191THENM=4 ELSEM=1
90 LINE(BX,BY)-(X,Y),PSET:IF PEEK
   (338)=191 AND A$="" THEN BX=X:
   BY=Y:BF=1:GOSUB5ØØ
100 IFPEEK(339)=V ANDLEN(A$)>Ø THEN
   16Ø
110 IFPEEK(341)=V ANDY-M>141 THEN
   Y=Y-M
120 IFPEEK(342)=V ANDY+M<191 THEN
   Y=Y+M
130 IFPEEK(343)=V ANDX-M>127 THEN
   X=X-M
140 IFPEEK(344)=V ANDX+M<223 THEN
   X=X+M
150 GOTO6Ø
160 GOSUB2ØØØ:SCREEN1,Ø:FORA=ØTO6:
   PCLS1:FORG=ØTOA:PRESET(Ø,G*2):NEXT
170 GOSUB1ØØØ
180 G$=INKEY$
190 IF A=Ø AND G$="" THEN18Ø
```





The firt step is to draw in the outline of your design

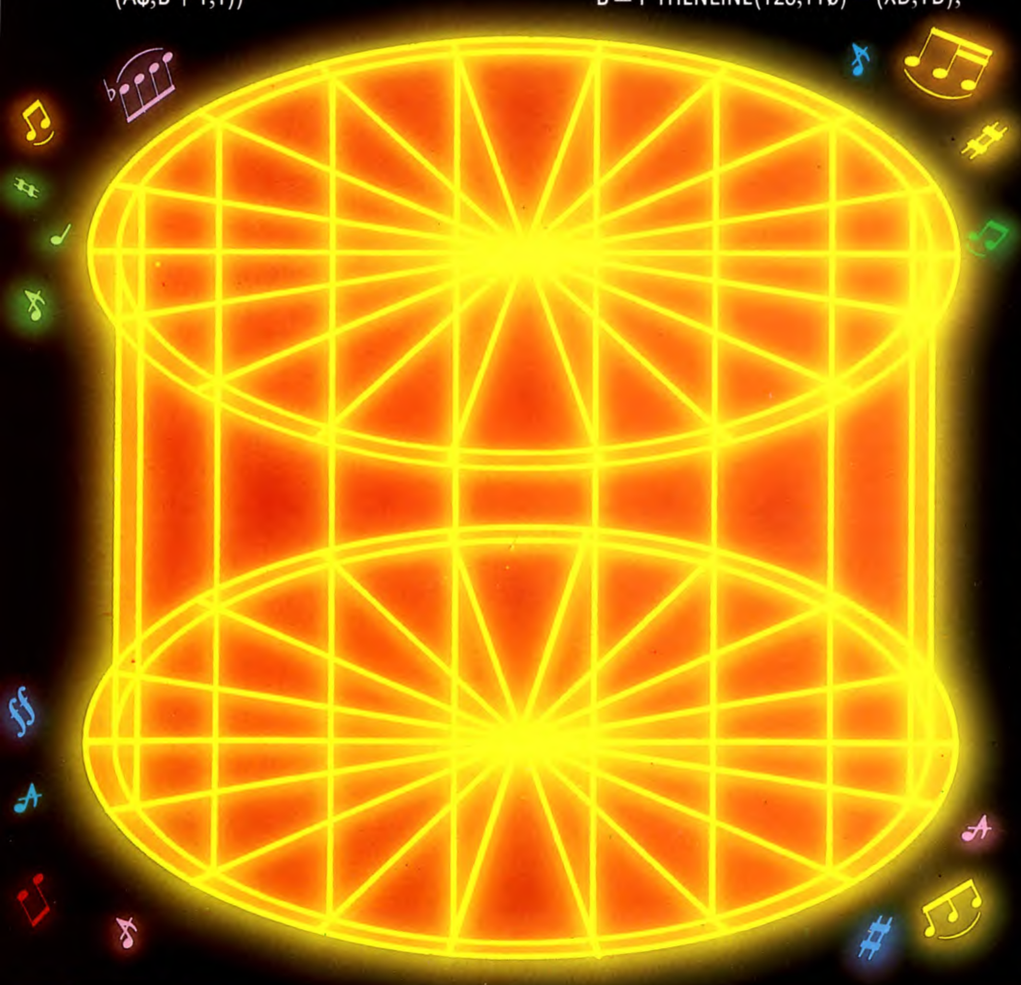


The computer then builds up the picture a section at a time



The finished design is a solid-looking wireframe drawing

```
200 IF A=Ø AND G$="□" THEN160
210 NEXT
220 FORA=ØTO6:PCOPY5+A*2TO1:PCOPY
    6+A*2TO2:NEXT
230 IFINKEY$="□" THEN160 ELSE220
500 LINE(BX,BY)-(X,Y),PRESET
510 LINE(255-BX,BY)-(255-X,Y),
    PRESET
520 BX=X:BY=Y:A$=A$+CHR$
    (X-128)+CHR$(190-Y):RETURN
1000 FORB=1TOLEN(A$) STEP2
1010 X=ASC(MID$(A$,B,1)):Y=ASC(MID$
    (A$,B+1,1))
1020 GOSUB1500:NEXT
1030 PCOPY1TO5+A*2:PCOPY2TO6+A*2
1040 RETURN
1500 FORC=ØTO360 STEP20
1510 D=C+20*A/7:PY=IS*Y
1520 GOSUB1530:NEXT:RETURN
1530 YD=110-SIN(D*RD)*X*COS(I*RD)-
    PY:XD=128+COS(D*RD)*X
1540 IFC=Ø THENLINE(XD,YD)-(XD,YD),
    PRESET:BX=XD:BY=YD
1550 LINE-(XD,YD),PRESET
1560 IFB=1 ANDBF=1 THEN1580 ELSEIF
    B=1 THENLINE(128,110)-(XD,YD),
    PRESET:GOTO1580
1570 LINE(A(C/20,Ø),A(C/20,1))-(XD,YD),
    PRESET
1580 A(C/20,Ø)=XD:A(C/20,1)=YD:
    RETURN
2000 CLS:INPUT"□ANGLE OF VIEW
    (Ø-180)□";I
2010 IF I<ØOR I>180 THEN2000
2020 IS=SIN(I*RD):RETURN
```



HOW IT WORKS

All the programs work in much the same way. The important parts are the routines that let you draw the outline, create the solid and cause it to spin.

ENTERING YOUR PROFILE

The rubber-banding routine starts at Line 14Ø on the Spectrum, Line 6Ø on the Commodore, Dragon and Tandy and PROCDRAW on the Acorns. This routine is mostly concerned with checking which keys have been pressed and updating the coordinates of the stretched line. The actual drawing of the line is done by the routine at Line 5ØØ, which also draws in the mirror image of the line.

As each line is fixed, the coordinates of its end point are stored. First of all a number is subtracted from each coordinate in order to give a point measured relative to the centre line. Then, on all but the Acorn, the numbers are turned into a string and each is added on to the end of A$ (or a$). On the Acorn, the coordinates are stored in two arrays X() and Y().

The routine at Line 2ØØØ (Line 7Ø on the Spectrum) lets you INPUT your angle of view; any angle between Ø and 18Ø degrees is accepted. The program now immediately jumps to the routine at Line 1ØØØ which creates and draws the three-dimensional solid.

THE THIRD DIMENSION

This routine uses the stored coordinates of the lines you have drawn and the angle you



INPUT to turn the outline into a solid-looking three-dimensional object.

The maths used by the routine is the same for each computer although the graphics commands to draw the lines are, of course, different.

In Line 1000 a loop, controlled by the variable B, selects each of your lines in turn and strips off their finishing coordinates from A$ (or the two arrays on the Acorn). It then jumps to the next routine starting at Line 1500.

This routine displaces these coordinates so they take into account the angle of view you selected, and then draws in the line using these displaced coordinates. This is repeated 20 times, so your original line is drawn in 20 different positions round a circle. The routine then RETURNs to Line 1000 to do the same for the next of your lines. By taking the lines one at a time the picture is built up in stages and you'll see this when you RUN the program.

If you look closely at the program you'll see that the procedure for drawing the first of your lines is slightly different from that for the others. This is because the displaced coordinates it works out are always the *ends* of the lines, the start coordinates being the ends of the previous lines. However, with the first line the start coordinates also have to be worked out.

Lines 1560 and 1565 check to see if it is the first line. The Acorn program works slightly differently, but on all the others B = 1 means the first line, and if BF = 1 as well it means the first line was a blank move. With a blank move all that happens is the coordinates are worked out and then stored in the array A(). These coordinates will then form the start of the next set of lines. If the first move wasn't blank it means it started from the centre, so 'spokes' are drawn out from the centre to the end coordinates of the lines. The Acorn assumes your first move was blank and works out all the coordinates even when you start from the centre.

So, for each line of your drawing, a whole ring of lines are drawn and their ends are linked by a circle, giving the effect of three dimensions. The coordinates of each end point are stored in the array A() so the computer knows the starting points for the next set of lines.

ROTATE

When the drawing is complete the program waits. Unless you have a Commodore, pressing a key will then call the paged graphics routine that makes the drawing spin. Several pictures are drawn one after the other and saved in memory. The number of pictures A, is controlled by the loop in Lines 610 to 690 on the Spectrum, Line 120 on the Acorn and Line 220 on the Dragon and Tandy. The variable A also determines the amount each picture is shifted by altering the angle in Line 1510. The pictures are then displayed one after the other to give the effect of animation.

MODELLING REALITY

Choosing the winning combination in a chance event is like picking numbers at random—a prime task for your micro, given a little statistical know-how



Engineers of all descriptions build models—structures in miniature—to test their designs. By doing so, they are simulating a real system. The article on pages 1158 to 1162 shows how your micro can do something similar—go through the motions of an event and give results that resemble those of an actual occurrence. Now you can find out how to combine the principles of statistics and modelling to solve problems realistically in games and real-life situations.

TYPES OF MODEL

Technically, a model is a representation of a real system, constructed for its simplicity and usefulness as an aid to making decisions. There are three categories of models—iconic, analogue and symbolic—not all of which are practicable for simulations by home computer enthusiasts.

Iconic models have no moving parts—road maps and toy aeroplanes come into this category. In preparation for building a bridge across an estuary, for example, civil engineers might build a model of the bridge and area surrounding the estuary to study the effects of tide, flooding and wind on the structure. Although computers are employed to monitor these factors and analyse the results, the fact that you have to collect data from an actual trial means that there is no need for a simulation.

Analogue models represent one quantity by another. For example, Professor Phillips, a British economist, arranged a number of cold-water tanks at different levels and used the flow of liquid between the tanks to represent the flow of cash in the UK economy. The same sort of thing can be done electronically, by representing different quantities by different voltages—but analogue computers, rather than the digital system used in home computers, are generally employed for this work. It is the third category—the *symbolic* or *abstract* model—that is of particular interest to anyone wishing to carry out simulations on a home computer.

REAL-LIFE UNCERTAINTIES

In the symbolic type of model, it is a mathematical formula that is used to represent the real system. For instance, if you wished to study the distance (S) travelled by a car at constant speed (V) for a Time (T), then a suitable model would be S = V*T. This type of equation is called *deterministic*, because it gives no measure of uncertainty. It describes precisely the motion of the car for any set of values you specify. There are, however, many events that cannot be specified without a degree of uncertainty. An example of these is in the program on pages 694 to 700, which simulates the tossing of a coin. Models that describe imprecise events are called *stochastic*. Usually, these are the types of event for which simulations are required—particularly in games.

Some variables can take only discrete values, such as 0, 1, 2, 3, ... The number of goals in a football match, or the number of male children in a family, are in this category. When simulating random variables of this type, it is important to be able to recognize that many seemingly different processes do, in fact, have the same underlying structure.

As an example, consider the coin tossing simulation mentioned earlier. This program could easily be modified to represent the random casts of dice or the number of goals scored in a penalty shoot-out. In all these circumstances, there are a number of independently repeated trials, the results of which can occur with definite probabilities. For example, a trial may consist of tossing a coin four times. The result of each trial can be four heads, three heads and a tail, two heads and two tails, three tails and a head or a complete set of four tails. And there is a definite probability attached to each of these outcomes. Structures of this type, named Bernoulli Processes, generate discrete random

- TYPES OF MODEL
- REAL-LIFE UNCERTAINTIES
- MODELLING THE PROBABILITY OF CHANCE BEHAVIOUR
- TYPES OF RANDOM VARIABLES
- SIMULATING RESULTS
- UNIFORM DISTRIBUTION
- EXPONENTIAL DISTRIBUTION
- NORMAL DISTRIBUTION
- COMPARING SIMULATED EVENTS



variables that are unsuitable for, say, the number of accidents occurring in a Grand Prix, the total number of telephone calls arriving in a half-hour period, or the number of lightning flashes recorded during an electric storm. In these instances, the events occur haphazardly during a period of time, of their own accord, and not as a result of repeated trials. Such structures are called Poisson Processes. Enter the first program, which lets you sample from any Poisson distribution you choose:

```
10 BORDER Ø:PAPER Ø:INK 7:CLS
20 POKE 23658,Ø
40 PRINT AT 1,6;INVERSE 1;"□POISSON
   SIMULATION□":PRINT:PRINT
50 INPUT "WHAT IS YOUR MEAN
   VALUE□";a
70 INPUT "SAMPLE SIZE□";n
80 FOR i=1 TO n
90 LET c=Ø:LET t=1
100 LET s=EXP (-a)
110 LET t=t*(RND*1)
120 IF t<=s THEN PRINT;"□□□□□□
   □";c,:GOTO 150
130 LET c=c+1
140 GOTO 110
150 NEXT i
160 INPUT "□ANOTHER SAMPLE
   (Y/N)?";g$
170 IF g$="y" THEN GOTO 10
180 STOP
```

```
40 PRINT "■■>■■POISSON
   SIMULATION■■"
50 PRINT"■WHAT IS YOUR MEAN
   VALUE":INPUT A
70 INPUT "■SAMPLE
   SIZE";N
80 FOR I=1 TO N
90 C=Ø:T=1
100 S=EXP(-A)
110 T=T*RND(1)
120 IF T<=S THEN PRINT C,:GOTO 150
130 C=C+1
140 GOTO 110
150 NEXT I:PRINT
160 PRINT"ANOTHER SAMPLE (Y/N)?"
170 GETG$:IF G$="Y" THEN RUN
180 IF G$<>"N" THEN 170
190 PRINT"■"
```

```
40 MODE1:PRINTTAB(12,3)"POISSON
   SIMULATION"''
50 INPUT"WHAT IS YOUR MEAN VALUE",A
70 INPUT'"AND YOUR SAMPLE SIZE",N
80 FOR I=1 TO N
90 C=Ø:T=1
100 S=EXP(-A)
110 T=T*RND(1)
120 IF T<=S THEN PRINT C;:GOTO 150
130 C=C+1
140 GOTO 110
150 NEXT
160 INPUT'"ANOTHER SAMPLE (Y/N)",G$
170 IF G$="Y" THEN 40
180 END
```

```
30 CLS
40 PRINT@7,"poisson simulation":PRINT:
   PRINT
50 INPUT"WHAT IS YOUR MEAN VALUE□";A
60 PRINT
70 INPUT"SAMPLE SIZE□";N
80 FOR I=1 TO N
90 C=Ø:T=1
100 S=EXP(-A)
110 T=T*RND(Ø)
120 IF T<=S THEN PRINTLEFT$(STR$(C)
   +"□□□□□□□□",8);:GOTO150
130 C=C+1
140 GOTO110
150 NEXT I
160 PRINT:INPUT"ANOTHER SAMPLE
   (Y/N)□";G$
170 IF G$="Y" THEN 30
180 END
```

When you RUN the program, you are prompted to enter a mean value (the mathematical average) and the size of your sample (Lines 5Ø and 7Ø). The main part of the program (Lines 8Ø to 15Ø) uses an equation to generate the Poisson variables. The variable S is set (Line 1ØØ) to the value of e (a mathematical constant) raised to the power A (the mean value you entered). Line 11Ø sets a random number between Ø and the value of T, and scales it according to the value of T. Line 12Ø then compares S and T to decide whether a variable (C) is to be printed.

Suppose, for example, you are designing a space game in which a battle cruiser has to pass through a meteorite storm. If the mean number of 'hits' in any one-minute period is two, and five separate one-minute simulations are required, the computer might print out 2, 3, 2, 1, Ø for 'hit' statistics. Each item in the list will not differ much from 2 (the mean), and there are five items—the sample size.

Could this same technique be used to simulate, say, the final scores in a football match? Clearly, you are dealing with a discrete variable, because the number of goals scored can take only whole-number values Ø, 1, 2, 3, and so on. Also, you are concerned with events that occur of their own accord

over time—and not as a result of repeated trials. So this is indeed a case for which a Poisson Process is suitable. Now, besides the number of matches played, the only other piece of information you need is what value is to be used as mean goals scored. Here, it becomes necessary to go out into the real world and collect some data. The table below gives mean scores for home and away teams in the English First Division soccer league, between 1977 and 1983.

First Division Goals

Season	Home goals per match	Away goals per match
77–78	1.6Ø39	1.Ø6Ø6
78–79	1.5498	1.Ø844
79–8Ø	1.6925	Ø.9481
8Ø–81	1.6364	1.Ø216
81–82	1.4545	1.Ø844
82–83	1.7532	Ø.9822

On the basis of these figures, an inspired guess of 1.7 for mean home scores and 1.Ø for mean away scores is reasonable. Dividing each value by two leads to mean half-time scores of Ø.85 and Ø.5. Construction of the program is now straightforward:

```
20 DIM a$(25,10):DIM h$(25,10):DIM
   h(50):DIM a(50):DIM f(50):DIM g(50)
30 BORDER 0:PAPER 0:INK 7:CLS
50 PRINT AT 0,8; INVERSE 1;
   "□□FINAL SCORE□□"
70 INPUT "□HOW MANY MATCHES (1–25)
   ?□";n
75 IF n<1 OR n>25 THEN GOTO 70
80 FOR i=1 TO n
90 PRINT "MATCH",i
100 INPUT "HOME TEAM";h$(i)
110 INPUT "AWAY TEAM";a$(i)
120 NEXT i
130 PAUSE 400
140 PRINT
150 FOR i=1 TO 2*n
160 LET c=0: LET t=1
170 LET s=EXP (−.85)
180 LET t=t*(RND*1)
190 IF t<=s THEN LET h(i)=c: GOTO 220
200 LET c=c+1
210 GOTO 180
220 NEXT i
230 FOR i=1 TO 2*n
240 LET c=0: LET t=1
250 LET s=EXP (−.5)
260 LET t=t*(RND*1)
270 IF t<=s THEN LET a(i)=c: GOTO 300
280 LET c=c+1
290 GOTO 260
300 NEXT i
310 CLS
320 PRINT TAB (7);"HALF-TIME SCORES";'
330 FOR i=1 TO n
340 PRINT h$(i);h(i);TAB 20;
   a$(i);a(i):PRINT
350 NEXT i
360 PRINT "PRESS ANY KEY TO CONTINUE"
370 IF INKEY$="" THEN GOTO 370
375 CLS
380 PRINT TAB 12;"FINAL SCORE"''
390 FOR i=1 TO n
400 LET f(i)=h(i)+h(n+i)
410 LET g(i)=a(i)+a(i+n)
420 PRINT h$(i);f(i);TAB 20;
   a$(i);g(i):PRINT430 NEXT i
440 INPUT " ANOYHER GO (y/n) ?□";g$
450 IF g$="n" THEN GOTO 490
460 INPUT "□SAME TEAMS (y/n)";p$
470 IF p$="n" THEN GOTO 70
480 GOTO 150
490 STOP
```

```
20 DIM A$(25),H$(25),H(50),A(50),FA(25),
   FH(25)
50 PRINT "■■>■■FINAL
   SCORE■■"
70 PRINT "HOW MANY MATCHES(1–25)":
   INPUT N
75 IF N<1 OR N>25 THEN GOTO 70
80 FOR I=1 TO N
90 PRINT "■MATCH",I
100 INPUT "HOME TEAM";H$(I)
110 INPUT "AWAY TEAM";A$(I):
   PRINT
120 NEXT I
130 FOR V=1 TO 1000:NEXT V
150 FOR I=1 TO 2*N
160 C=0:T=1
170 S=EXP(−.85)
180 T=T*RND(1)
190 IF T<=S THEN H(I)=C:
   GOTO 220
200 C=C+1
210 GOTO 180
220 NEXT I
230 FOR I=1 TO 2*N
240 C=0:T=1
250 S=EXP(−.5)
260 T=T*RND(1)
270 IF T<=S THEN A(I)=C:
   GOTO 300
280 C=C+1
290 GOTO 260
300 NEXT I
320 PRINT "■■>■■HALF-TIME
   SCORES■■"
330 FOR I=1 TO N
340 PRINT H$(I);H(I):PRINT"■";
   A$(I);A(I):PRINT
350 FOR D=1 TO 200:NEXT'D,I
360 PRINT "PRESS ANY KEY TO CONTINUE"
370 GET X$:IF X$="" THEN 370
380 PRINT "■■>■■FINAL
   SCORE■■"
390 FOR I=1 TO N
400 FH(I)=H(I)+H(N+1)
410 FA(I)=A(I)+A(I+N)
420 PRINT H$(I);FH(I):PRINT"■";A$(I);
   FA(I):PRINT
430 FORD=1TO200:NEXT D,I
440 PRINT "■ANOTHER GO (Y/N)?"
450 GETG$:IF G$="N" THEN
   PRINT"■":END
455 IF G$<>"Y" THEN 450
460 INPUT "■SAME TEAMS (Y/N)";P$
470 IF P$="N" THEN RUN
480 GOTO 150
```

```
20 DIM A$(25),H$(25),H(50),A(50),FA(25),
   FH(25)
50 MODE1:PRINTTAB(16,3)"FINAL SCORE"
60 PRINT:PRINT
70 INPUT"HOW MANY MATCHES (1–25)",N
75 IF N<1 OR N>25 THEN 70
80 FOR I=1 TO N
90 PRINT"MATCH", I
100 INPUT"HOME TEAM",H$(I)
110 INPUT"AWAY TEAM",A$(I):PRINT
```



```
120 NEXT
130 D=INKEY(1ØØ)
140 PRINT
150 FOR I=1 TO 2*N
160 C=Ø:T=1
170 S=EXP(-.85)
180 T=T*RND(1)
190 IF T<=S THEN H(I)=C:
GOTO 22Ø
200 C=C+1
210 GOTO 18Ø
220 NEXT
230 FOR I=1 TO 2*N
240 C=Ø:T=1
250 S=EXP(-.5)
260 T=T*RND(1)
270 IF T<=S THEN A(I)=C:
  GOTO 3ØØ
280 C=C+1
290 GOTO 26Ø
300 NEXT
320 CLS:PRINTTAB(12,3)"HALF-TIME
  SCORES":PRINT:PRINT
330 FOR I=1 TO N
340 PRINTH$(I)H(I)TAB(27)A$(I)A(I)
350 NEXT
360 PRINT"PRESS ANY KEY TO CONTINUE"
370 X$=GET$
375 CLS
380 PRINTTAB(16,3)"FINAL SCORE":
  PRINT:PRINT
390 FOR I=1 TO N
400 FH(I)=H(I)+H(N+1)
410 FA(I)=A(I)+A(I+N)
420 PRINTH$(I)FH(I)TAB(27)
  A$(I)FA(I):PRINT
430 NEXT
440 INPUT"ANOTHER GO (Y/N)",G$
450 IF G$="N" THEN END
460 INPUT"SAME TEAMS (Y/N)",P$
470 IF P$="N" THEN 7Ø
480 GOTO 15Ø
```

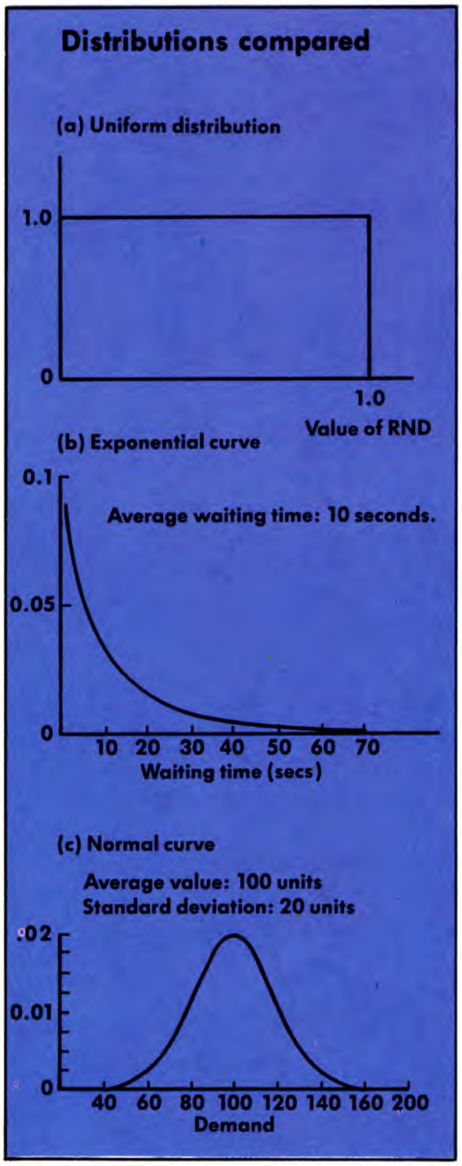
```
20 DIM A$(25),H$(25),H(5Ø),A(5Ø),FA(25),
  FH(25)
30 CLS
50 PRINT@11,"final score"
60 PRINT:PRINT
70 INPUT"HOW MANY MATCHES(1-25)□";
  N
75 IF N<1 OR N>25 THEN 7Ø
80 FOR I=1 TO N
90 PRINT"MATCH□";I
100 INPUT"HOME TEAM□";H$(I)
110 INPUT"AWAY TEAM□";A$(I):
  PRINT
120 NEXT I
130 FOR V=1 TO 2ØØØ:NEXT V
140 PRINT
150 FOR I=1 TO 2*N
160 C=Ø:T=1
170 S=EXP(-.85)
180 T=T*RND(Ø)
190 IF T<=S THEN H(I)=C:
  GOTO 22Ø
200 C=C+1
210 GOTO 18Ø
220 NEXT I
230 FOR I=1 TO 2*N
240 C=Ø:T=1
250 S=EXP(-.5)
260 T=T*RND(Ø)
270 IF T<=S THEN A(I)=C: GOTO 3ØØ
280 C=C+1
290 GOTO 26Ø
300 NEXT I
310 CLS
320 PRINT@7,"half-time scores":
  PRINT:PRINT
330 FOR I=1 TO N
340 PRINTH$(I);H(I),A$(I);A(I)
350 NEXT I
360 PRINT"PRESS ANY KEY TO CONTINUE"
370 IF INKEY$="" THEN 37Ø
375 CLS
380 PRINT@11,"final score":
  PRINT:PRINT
390 FOR I=1 TO N
400 FH(I)=H(I)+H(N+I)
410 FA(I)=A(I)+A(N+I)
420 PRINTH$(I);FH(I),A$(I);
  FA(I)
430 NEXT
440 PRINT:INPUT"ANOTHER GO
  (Y/N)□";G$
450 IF G$="N" THEN END
460 INPUT"SAME TEAMS (Y/N)□";P$
470 IF P$="N" THEN 7Ø
480 GOTO 15Ø
```



RUN the program and respond to the prompts of the first section (from start to Line 12Ø). To avoid too much typing, you can input letters of the alphabet, instead of actual names, for both teams. Lines 15Ø to 3ØØ use the Poisson algorithm, used in the first program, to generate half-match scores for the Home and Away sides. Lines 32Ø to 35Ø organize the printing of half-time scores and Lines 39Ø to 43Ø the display of full-time results.

Although the simulated *individual* goals per match won't tally well with the actual results, the simulated *aggregate* performance should resemble the actual results closely.

The Poisson distribution works well for whole-number variables, but some variables are fractional—they are continuously random. In a computer simulation of the Olympic long jump, for example, a competitor might record any distance between seven



and nine metres—the length of the jump is an example of a continuous random variable.

Although the computer's RND function is suitable for simulating Poisson-type whole-number variables, you need a modified RND function to model with continuous variables. Figure 1 compares the type of distribution given by both types of variables. Whole-number variables give a *uniform* distribution, but continuous variables can give either an *exponential* or a *normal* distribution. Enter the next program to transform the flat or uniformly distributed random variables supplied by the computer's RND function into exponentially distributed random variables:

```
10 BORDER 0:INK 7PAPER 0:CLS
20 POKE 23658,0
40 PRINT AT 0,4; INVERSE 1;
   "□EXPONENTIAL SIMULATION□""
50 INPUT "WHAT IS YOUR MEAN VALUE
   ?□"));a
70 INPUT "□SAMPLE SIZE ?□";n
80 FOR i=1 TO n
90 LET x=(-a)*LN (RND*1):LET y=INT
   (x*10):PRINT "□□□□□□";.1*y,
100 NEXT i
110 INPUT "□ANOTHER SIMULATION
   (y/n)□";g$
120 IF g$="y" THEN GOTO 30
130 STOP
```

```
40 PRINT "■■>■■EXPONENTIAL":
   PRINT"■■■SIMULATION□■■"
50 PRINT "WHAT IS YOUR MEAN":INPUT
   "VALUE";A
70 INPUT "■■SAMPLE SIZE";N
80 FOR I=1 TO N
90 X=(-A)*LOG(RND(1)):Y=INT(X*10):
   PRINT .1*Y,
100 NEXT I
110 PRINT:PRINT "ANOTHER SAMPLE
   (Y/N)?"
120 GETG$:IF G$="Y" THEN RUN
130 IF G$<>"N" THEN 120
140 PRINT"■"
```

```
40 MODE1:PRINTTAB(11,3)"EXPONENTIAL
   SIMULATION"
50 INPUT'"WHAT IS MEAN VALUE",A
70 INPUT'"SAMPLE SIZE",N
80 FOR I=1 TO N
90 X=-A*LOG(RND(1)):Y=INT(X*10):
   PRINT.1*Y;
100 NEXT
110 INPUT'"ANOTHER SIMULATION",G$
120 IF G$="Y" THEN 40
```

```
30 CLS
40 PRINT@5,"exponential simulation":PRINT:
   PRINT
50 INPUT"WHAT IS YOUR MEAN VALUE□";
   A
60 PRINT
70 INPUT"SAMPLE SIZE□";N
80 FOR I=1 TO N
90 X=(-A)*LOG(RND(0)):Y=INT(X*10):
   PRINTLEFT$(STR$(Y/10)+"□□□□□
   □□",8);
100 NEXT I
110 PRINT:INPUT"ANOTHER SIMULATION
   (Y/N)□";G$
120 IF G$="Y" THEN 30
130 END
```

When you RUN the program, specify a mean value and the number of samples required, then the variables will be printed on the screen. Line 90 does all of the work. It uses a mathematical technique, called the inverse transform method, to translate 'flat' variables into exponential variables. This program could be used equally well to simulate, say, the waiting time for driers in a launderette, or the lifetimes of energy packs in a space adventure game. Time elapsed is the important concept in both cases.

NORMAL DISTRIBUTION

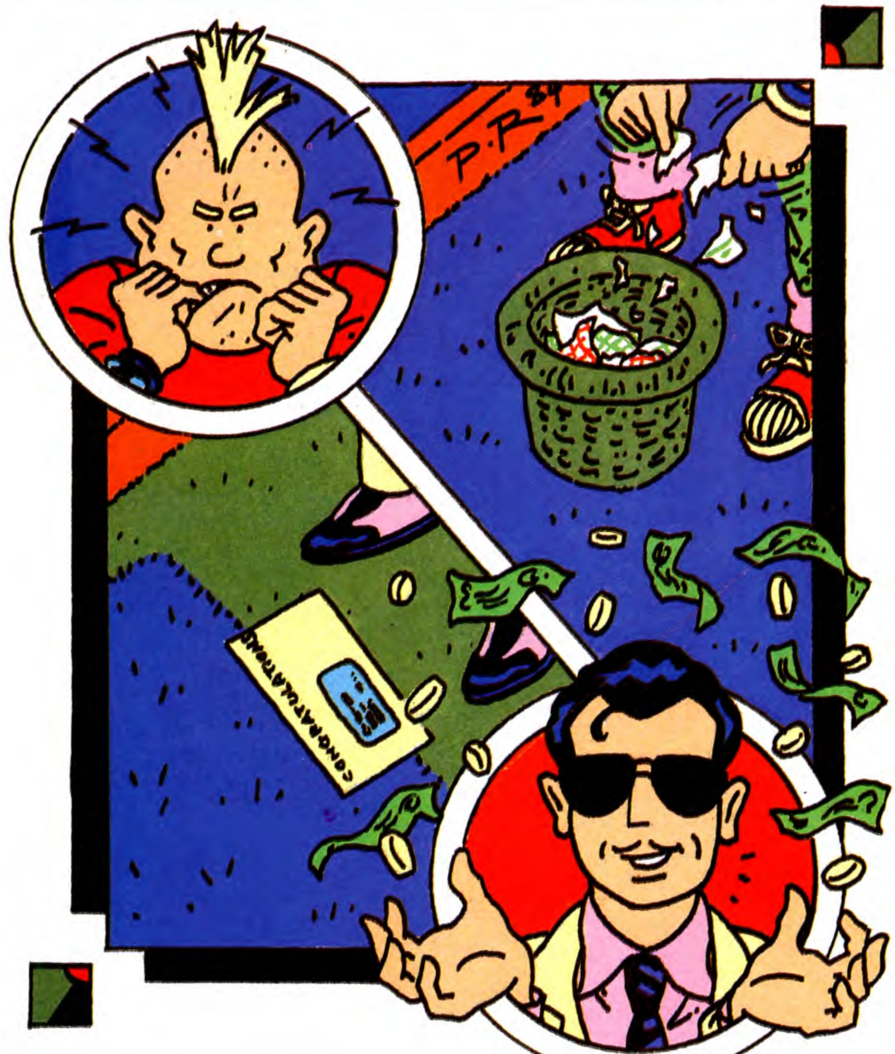
The normal or bell-shaped curve represents the most famous statistical distribution of all. It has been found useful in describing all sorts of natural phenomena, including the heights and weights of adults. In addition, it is widely accepted as the most appropriate means of describing error distributions (see page 697).

As an illustration of how the normal distribution might be incorporated into a model, consider the following example to design a business game. Suppose company sales (S) depend on some base level (S_0) and the amount spent on advertising (A). The relationship could be summarized by the expression $S=S_0+(B*A)$ where B is some known constant.

Now in real life, such precise relationships rarely hold. Unforseen events or circumstances distort the facts or introduce 'noise' into the system. This system can be adequately modelled by introducing a normally distributed disturbance equation for sales. The complete equation for sales would then be $S=S_0+(B*A)+\text{Normal Disturbance}$.

Enter the next program to see this work:

```
10 BORDER 0:PAPER 0:INK 7:CLS
40 DIM u(50):DIM y(50):DIM z(50)
50 PRINT AT 0,7;INVERSE 1;"□NORMAL
   SIMULATION□""
60 INPUT "□WHAT IS YOUR MEAN VALUE
   ?□";a
70 INPUT "□WHAT IS YOUR MINIMUM
   VALUE ?□";m
80 INPUT "□SAMPLE SIZE ?□";n
90 LET b=(a-m)/2.5
100 FOR i=1 TO n
110 LET t=0
120 FOR j=1 TO 15
130 LET y(j)=RND*1
140 LET t=t+y(j): NEXT j
150 LET u(i)=((t/15)-.5)/SQR (1/(12*15))
160 LET z(i)=a+(b*u(i))
170 LET p=INT (10*z(i)): PRINT
   "□□□□□□";.1*p,
180 NEXT i
190 INPUT "□ANOTHER SAMPLE ?□";g$
200 IF g$="y" THEN GOTO 50
210 STOP
```




```
40 DIM U(5Ø),Y(5Ø),Z(5Ø)
50 PRINT "■■>■■NORMAL
   SIMULATION■■"
60 PRINT"WHAT IS YOUR MEAN":INPUT
   "VALUE";A
70 PRINT"■WHAT IS YOUR MINIMUM":
   INPUT"VALUE";M
80 INPUT "■SAMPLE SIZE";N
90 B=(A-M)/2.5
100 FOR I=1 TO N
110 T=Ø
120 FOR J=1 TO 15
130 Y(J)=RND(1)
140 T=T+Y(J):NEXT J
150 U(I)=((T/15)-.5)/SQR(1/(12*15))
160 Z(I)=A+(B*U(I))
170 P=INT(1Ø*Z(I)):PRINT .1*P,
180 NEXT I
190 PRINT:PRINT "ANOTHER SAMPLE
    (Y/N)?"
200 GETG$:IF G$="Y" THEN RUN
210 IF G$<>"N" THEN 200
220 PRINT"■"
```

```
40 DIM U(15Ø),Y(15Ø),Z(15Ø)
50 MODE1:PRINTTAB(12,3)"NORMAL
   SIMULATION"''
60 INPUT"WHAT IS YOUR
   MEAN VALUE",A
70 INPUT"WHAT IS YOUR MINIMUM
   VALUE",M
80 INPUT"SAMPLE SIZE",N
90 B=(A-M)/2.5
100 FOR I=1 TO N
110 T=Ø
120 FOR J=1 TO 15
130 Y(J)=RND(1)
140 T=T+Y(J):NEXT
150 U(I)=((T/15)-.5)/SQR
    (1/(12*15))
160 Z(I)=A+(B*U(I))
170 P=INT(1Ø*Z(I)):PRINT
    .1*P;
180 NEXT
190 INPUT'"ANOTHER
    SAMPLE",G$
200 IF G$="Y" THEN 50
```

```
30 CLS
40 DIM U(5Ø),Y(5Ø),Z(5Ø)
50 PRINT@7,"normal simulation":
   PRINT:PRINT
60 INPUT"WHAT IS YOUR MEAN
   VALUE□";A
70 INPUT"WHAT IS YOUR MINIMUM
   VALUE□";M
80 INPUT"SAMPLE SIZE□";N
90 B=(A-M)/2.5
100 FOR I=1 TO N
110 T=Ø
120 FOR J=1 TO 15
130 Y(J)=RND(Ø)
140 T=T+Y(J):NEXT J
150 U(I)=((T/15)-.5)/SQR (1/(12*15))
160 Z(I)=A+(B*U(I))
170 P=INT(1Ø*Z(I)):PRINTLEFT$(STR$
    (P/1Ø)+"□□□□□□□□",8);
180 NEXT I
190 PRINT:INPUT"ANOTHER SAMPLE□";G$
200 IF G$="Y" THEN 50
210 END
```

RUN the simulation, then specify the mean and minimum values of your variable. Technically, the range of the normal curve is unbounded, so there is a small chance that a simulated value will be less than the minimum you have specified. This won't happen very often.

Lines 12Ø to 15Ø use the RND function to generate 15 random numbers between Ø and 1, and then add the numbers together. Line 16Ø calculates the mean and adds a scaling factor. After further scaling (Line 16Ø), the result can be printed.

It may help you to get a better 'feel' for what's going on if numbers are generated from two distributions with the same mean and roughly the same range, but with different shapes. Run this program and enter values of 1ØØ for mean, 5Ø for minimum and 4Ø for sample size. The values you get could be plotted to give a graph that looks like figure 1C.

Now delete Line 9Ø and Lines 12Ø to 16Ø, and overwrite the following line (change RND(1) to RND(Ø) on the Dragon and Tandy):

```
110 Z(I)=M+RND(1)*2*(A-M)
```

These changes convert the program into one capable of printing out uniform random values. Run the program again, entering the same values as before, and compare the results. This time, the values could be plotted to give a graph that looks like figure 1A. You should notice that the normal readings (in the first test) are much more closely clustered about the mean value of.

CLIFFHANGER: ADDING SEAGULLS

While Willie struggles manfully with his dangerous task, gulls crane and wheel in the sky above adding a touch that can turn an imaginative idea into a viable product

No seaside scene would be complete without the plaintive gulls. But the game's own sound effects—coming in later parts—precluded this. So instead the seagulls are just going to flap around in the sky over the slope as part of the background scenery.

You might think that this is an unnecessary frill as it plays no real part in the game. But it is touches like this one that make all the difference if you are writing a commercial game. The days of two cursors and a blip tennis games are over.

Unfortunately, there are no flapping seagulls in the Dragon version of Cliffhanger. It would have burdened the game with even more DATA. Dragon owners have probably had all the DATA they can take.

Besides, in the Dragon version, it is a very hot day, much too hot for sensible seagulls to be flapping about. They are all resting in the shade!

The following program not only prints the gull up on the screen, it makes the gull flap its wings up and down by using simply two-frame animation.

```
        org 58751
gul     ld a,(57349)
        inc a
        res 3,a
        ld (57349),a
        ld bc,57192
        cp 4
        jr c,gpt
        ld bc,57208
gpt     ld a,46
        ld hl,42
        push bc
        call print
        inc hl
        call print
        pop bc
        ld hl,68
        call print
        inc hl
        call print
        ret
        org 58217
print   *
```

WINGING IT

Memory location 57,349 contains the so-called gull delay variable. This stops the wings flapping up and down too fast. The full delay is loaded into A and incremented.

To get the wings to flap up and down, the value of gull delay needs to fluctuate in a restricted range. Here the instruction res 3,a resets bit 3 of the accumulator. So when the gull variable is incremented beyond 7 it is set back to zero again.

The result of the increment and reset is stored back in 57,349.

To make the gull appear to be flapping its wings, there is data for two gulls—one with its wings up and one with its wings down. The two sets of data for these two gulls start at 57,192 and 57,208. Each comprises 16 bytes of data.

BC is loaded with the address of the first of these two blocks of data. Then the contents of the accumulator—which are still the gull delay—are compared with 4.

A cp is an unstored subtraction. The number 4 is taken away from the contents of the accumulator, but the result is not stored in the accumulator—or anywhere else for that matter. The only result of the operation is to set the flags.

So if the gull delay in the accumulator is less than 4—that is, three or below—the carry flag is set. And if the gull delay is 4 or over, the carry flag is not set.

The jr c instruction makes the processor jump if the carry flag is set. So if the gull delay is less than four, the processor skips the next instruction and 57,192 remains in the BC register. But if the gull delay is 4 or more, the carry flag is not set, the jump is not made and BC is loaded with 57,208, the beginning of the data for the second gull in the data table.



- DIRECTION CHECKING
- IDENTIFYING THE END OF THE FLIGHT PATH
- CHANGING DIRECTION
- FLIPPING IN THE FLAG

TAKING FLIGHT

A is loaded with 46, to set the gull colour, and HL is loaded with the position of the first gull on the screen, 42.

The contents of the BC register is then pushed onto the stack to preserve it. To add to the confusion, there are two gulls on the screen as well as in the data table. But the two gulls on the screen are the same gull in the data table. The same image is printed on the screen twice, in two different places. As the same image is used, the gulls flap in concert. So the data pointer has to be preserved while the first gull is being printed on the screen, as the print routine automatically updates the pointer while it prints out the sixteen bytes of data needed to make up a complete gull.

The print routine is then called. This prints out the eight bytes of data following the start position given in BC, updating the contents of BC as it goes. HL is then incremented to move it onto the screen position immediately to the right of the part of the gull already printed up. Then the print routine is called again.

When those two characters squares have been printed out, you have one complete gull. Then it is time to print the next one. So the data pointer is pulled off the stack, HL is moved onto the position of the second gull. Then the print routine is called again, HL is incremented and the print routine is called yet again to print up the second half of the second game.

This little routine makes the gulls flap their wings:

```
     ORG 22Ø96
     LDX #Ø
     LDY #Ø
LOOP LDA $C35Ø,X
     STA $FB
     LDA $C35A,X
     STA $FC
     LDA ($FB),Y
     CMP #61
     BNE XX
     LDA #57
     STA ($FB),Y
XX   LDA ($FB),Y
     CLC
     ADC #1
     STA ($FB),Y
     INX
     CPX #2
     BNE LOOP
     RTS
```

There are two gulls on the screen, and four gulls in memory. The two gulls on the screen are really the same gull printed in different places. And the four gulls in memory are the same gull pictured with its wings in different positions. So when the four gull pictures are printed on the same place on the screen, one after another, you get the impression that the gull is flying.

GULLING

The X register is used to count the number of gulls on the screen and is initialized to Ø. The Y register is used as the offset when indirect addressing is used later in the routine. But the offset is not used at this point and the contents of Y remain Ø throughout.

The low byte of the screen position of the first gull is stored in the variable table in memory location $C35Ø and the high byte is in $C35A. And the low and the high byte of the screen position of the second gull are stored in $C351 and $C35B. These screen positions were worked out and stored in these memory locations by the routine given in part seven of Cliffhanger (pages 11Ø1 to 11Ø5).



So the low byte of the screen position of the first gull is loaded into the accumulator and stored temporarily in the zero page location $FB where it can be manipulated easily. The high byte is stored at $FC. These two zero-page memory locations now act as a screen-position pointer.

LDA ($FB),Y loads the accumulator with the contents of the memory location pointed to by the contents of $FB and $FC—in other words, it loads up the contents of that particular position on the screen.

The four gulls are in memory in the form of UDGs. The data for these was entered as data in an earlier part of Cliffhanger. The gulls occupy UDGs 58 to 61 inclusively.

So when the accumulator is loaded with the contents of the screen position of the first gull, it's loaded with a number between 58 and 61—a gull with its wings in some position or other that has already been printed.

The number loaded is compared to 61. If it is not 61, the BNE instruction will skip over the next instructions which load up the accumulator with 57. This is 58 − 1 as the gull UDG to be incremented before it is printed on the screen. So if the gull UDG has already reached its maximum, 61, it is reset to 57 and passed back into the routine via the screen.

The UDG number in the accumulator is now between 57 and 60. The carry flag is cleared and 1 is added to the contents of the accumulator. The UDG bearing the resulting number is stored back on the screen in the same position by the instruction STA ($FB),Y.

So all this routine does is look at the gull on the screen and move it onto the next frame. And if it has reached the end of the four frames, it starts it off at the beginning again.

The X gull-counter is then incremented and compared to 2. It is less than two, only the first gull has been moved onto the next frame, so the processor loops back and goes through exactly the same process with the second one. And if X has been clocked up to 2, the processor exits the routine.

This program prints and moves two seagulls. In fact, these seagulls are much more ambitious than those on the other machines. As well as flapping their wings, they fly about. Don't forget to set up the computer as normal before you key it in.

```
70 DATA5,18,3,3,
   245,246,4
80 FORA%=&1
   C01TO&1C07:
   READ?A%:NEXT
120 FORPASS=
   0TO3STEP3
130 P%=&1C08
140 [OPTPASS
150 .Seagull
160 JSRLb1
170 LDA&85
180 AND#&40
190 BNELb6
200 INC&85
210 LDA&85
220 AND#&3F
230 CMP#23
240 BNELb7
250 LDA&85
260 ORA#&40
270 STA&85
280 JMPLb7
290 .Lb6
300 DEC&85
310 LDA&85
320 AND#&3F
330 CMP#0
340 BNELb7
350 LDA&85
360 AND#&BF
370 STA&85
380 .Lb7
390 LDA&86
400 AND#&40
410 BNELb8
420 INC&86
430 LDA&86
440 AND#&3F
450 CMP#30
460 BNELb9
470 LDA&86
480 ORA#&40
490 STA&86
500 JMPLb9
510 .Lb8
520 DEC&86
530 LDA&86
540 AND#&3F
550 CMP#10
560 BNELb9
570 LDA&86
580 AND#&BF
590 STA&86
600 .Lb9
610 LDA&85
620 EOR#&80
630 STA&85
640 JSRLb1
650 RTS
660 .Lb1
670 LDA&85
680 AND #&80
690 BNELb2
700 LDY#21
710 JMPLb3
720 .Lb2
730 LDY#23
740 .Lb3
750 LDX#245
760 TYA
770 PHA
780 JSR&17D4
790 LDX#246
800 PLA
810 TAY
820 INY
830 JSR&17D4
840 LDY#48
850 LDA&85
860 AND#&3F
870 TAX
880 JSR&1964
890 LDX#0
900 .Lb4
910 LDA&1C01,X
920 JSR&FFEE
930 INX
940 CPX#7
950 BNELb4
960 LDY#56
970 LDA&86
980 AND#&3F
990 TAX
```



```
1000 JSR&1964
1010 LDX#0
1020 .Lb5
1030 LDA&1C01,X
1040 JSR&FFEE
1050 INX
1060 CPX#7
1070 BNELb5
1080 RTS
1090 ]NEXT
```

To make the gulls fly, you have to set some initialization values by POKEing a couple of the variables on the zero page. You do this by keying in the following instructions:

```
?&85=0:?86=10
```

Then, when you have the rest of the game in memory, key in:

```
CALL &1B32:REPEAT CALL &1C08:FOR
A%=0 TO 500:NEXT:UNTIL 0
```

This should send the feathers flying.

LOWER THE FLAPS

The DATA in Line 70 define the characters that make up the gulls and their colour. This is READ into a data table at &1C01 to &1C07 where the machine code can find it.

The first thing the assembly language routine does is jump to the subroutine at the label Lb1. This is the routine that does the printing on the screen. Here, it is called to print blank spaces over the old seagulls on the screen so that the new ones can be printed up without leaving bits of the old ones behind. Then the contents of &85 are loaded into the accumulator and ANDed with &40. This looks at bit six. &85 is one of the zero page locations that controls the gulls. If bit six is set the gull is moving to the left, and if it is not set the gull is moving to the right.

ANDing with &40 gives a 1 if bit six is set and a 0 if bit six is not set. Then the BNE instruction in Line 190 jumps to the routine which moves the gull to the left starting at Line 290 if the result is 1. Or the processor continues with the routine that moves the gull to the right if the result is 0.

ALL RIGHT

The routine that moves the gull to the right does that by simply incrementing the gull control variable in &85. This byte is then loaded into the accumulator and ANDed with &3F. This examines bits zero to five, to see whether the gull has reached the end of its path. The ANDing clears bits six and seven, leaving the others unchanged. The result is compared to 23, which marks the extreme right-hand end of the gull's flight.

If the result has not clocked up to 23 yet, the gull has not got to the end of its path and it is not time to change direction. So the BNE instruction branches the processor over the next little routine.

If the result has clocked up to 23, the gull has reached the end of its path, so the processor does not make the branch. So the contents of &85 are loaded into the accumulator, ORed with &40 to set bit six and the result is stored back in &85.

Whichever way the gull is going the processor now jumps onto Lb7 in Line 380, to skip the routine that sends the gull to the left.

LEFT OVER

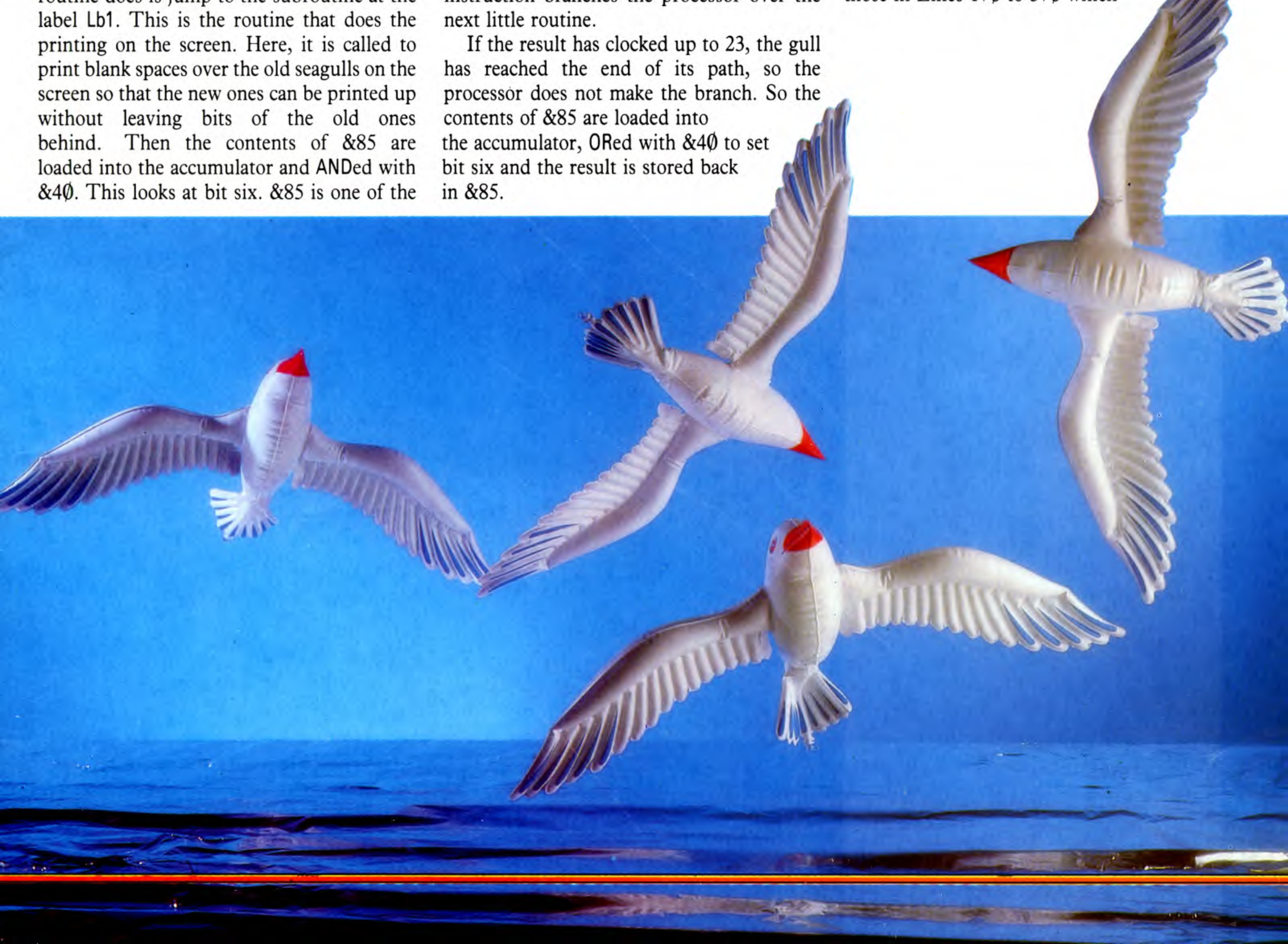
If the branch in Line 190 had decided that the gull was flying to the left it would have sent the processor straight to the instruction in Line 290.

The routine that moves the gull to the left works much like the one that moves it to the right. Only now, the control variable in &85 is decremented instead of being incremented.

Again the result is ANDed with &3F so that bits zero to five can be examined. The left-hand-most end of the gull's path is at 0. If the gull byte has clocked down that far the direction is reset by ANDing the contents of &85 with &BF, which resets bit six and leaves the rest as they were.

GULL TWO

The second gull is handled in exactly the same way. The instruction from Line 390 to Line 590 are exactly the same as those in Lines 170 to 370 which



move the first gull.

The only difference is that the control byte is &86 instead of &85. Otherwise, it moves the gull to the left or the right and checks to see whether it has reached the end of its path in exactly the same way.

THE FLAPS

The flapping of the wings of both gulls are controlled by bit seven of &85. The contents of this control byte are loaded into the accumulator and Exclusively ORed with &8Ø. This flips bit seven—if it was 1 bit become Ø, if it was Ø it becomes one. There are two frames for each gull, one with its wings up and one with its wings down. Once one has been printed, the other has to be printed next time. And the easiest way to control this is to flip a bit. The result is stored back in &85.

The processor then jumps to the subroutine which prints the new gull up. And when it comes back again, it returns.

NEW GULLS

The gull print routine, which starts at Line 87Ø, begins by loading up the first gulls control byte and ANDs it with &8Ø. This checks bit seven to see whether the wings should be up or down.

The branch instruction in Line 69Ø then either lets the processor move onto the next instruction which loads Y with 21, or skips that and loads Y with 23 which specifies the graphic number to be used.

Whichever graphic is going to be used, the processor ends up at the instruction in Line 75Ø. LDX #245 loads up the number of the character that is going to be redefined to make the gull flap its wings.

The graphic number in Y is transferred into A, so that it can be preserved by pushing it onto the stack by the PHA instruction.

The processor then jumps to the subroutine at &17D4 which is the routine which redefines a character. In this case, by redefining 245, the left-hand side of the gull is redrawn.

X is then loaded with 246. This is the number of the character used to form the right-hand side of the gull. The graphic number is then pulled off the stack, transferred back into the Y register and incremented. This moves the pointer onto graphics 22 and 24, which are going to be used to redefine the right-hand side of gull. The subroutine at &17D4 is called again, which redraws the right-hand side of the gull.

Y is then loaded with 48 which is the Y coordinate of the first gull. The X coordinate is picked up from the first gull's control byte in &85. It was put there—in bits zero to five—by the move routine above, remember.

The X coordinate of the first gull is recovered by the contents of &85 into the accumulator, ANDing with &3F again and transferring the result into the X register.

The processor then jumps to the subroutine at &1964, which moves the cursor into the appropriate position. X is then loaded with Ø and the processor moves into the routine which actually prints the gull on the screen.

GULLS

LDA &1CØ1,X loads up the first byte of the gull data table. The &FFEE routine is then called, which outputs the byte to the screen. X is incremented and compared with 7 and the processor branches back to output the next byte of data. Byte character numbers 245 and 246—the characters for the left and right-hand sides of the gulls—appear in the data.

Once all seven bytes of data have been output, the processor drops out of the loop. Y is then loaded with 56, the Y coordinate of the second gull. The X coordinate, again, is picked up from the control byte, in this case &86. Again, it has to be ANDed with &3F and transferred into the X register.

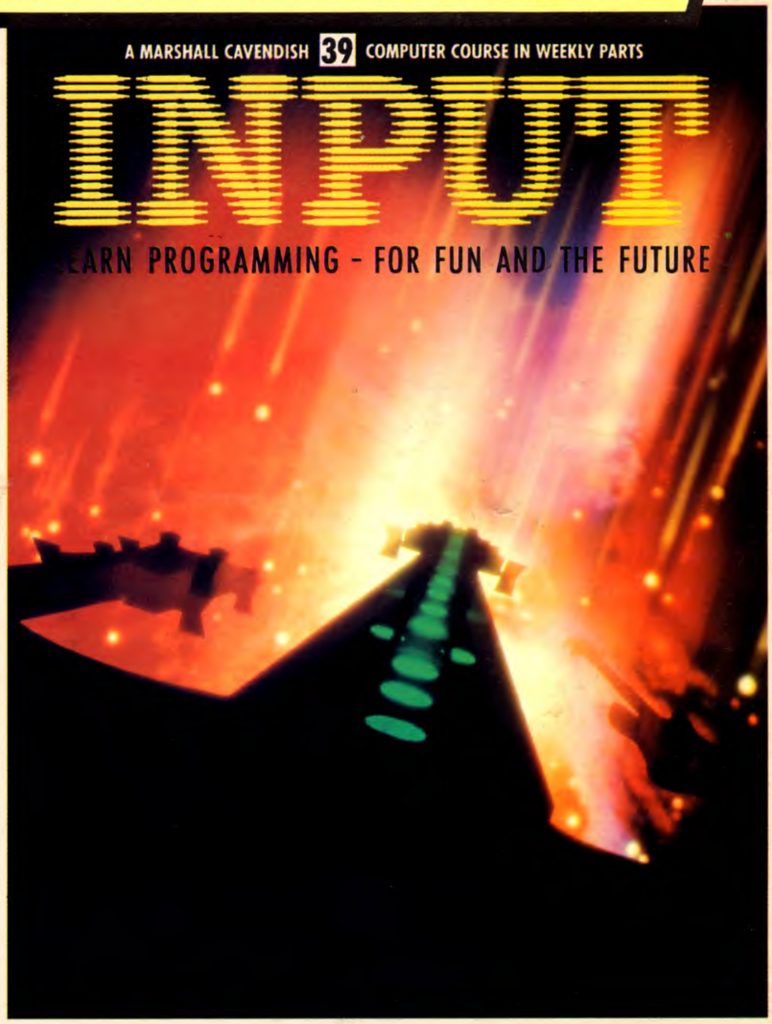
The instructions in Lines 1ØØØ to 1Ø7Ø are exactly the same as those in Lines 88Ø to 95Ø. They position the cursor and print out the data on the screen. And when all seven bytes of the gull data have been output in the second gull position, the processor returns.



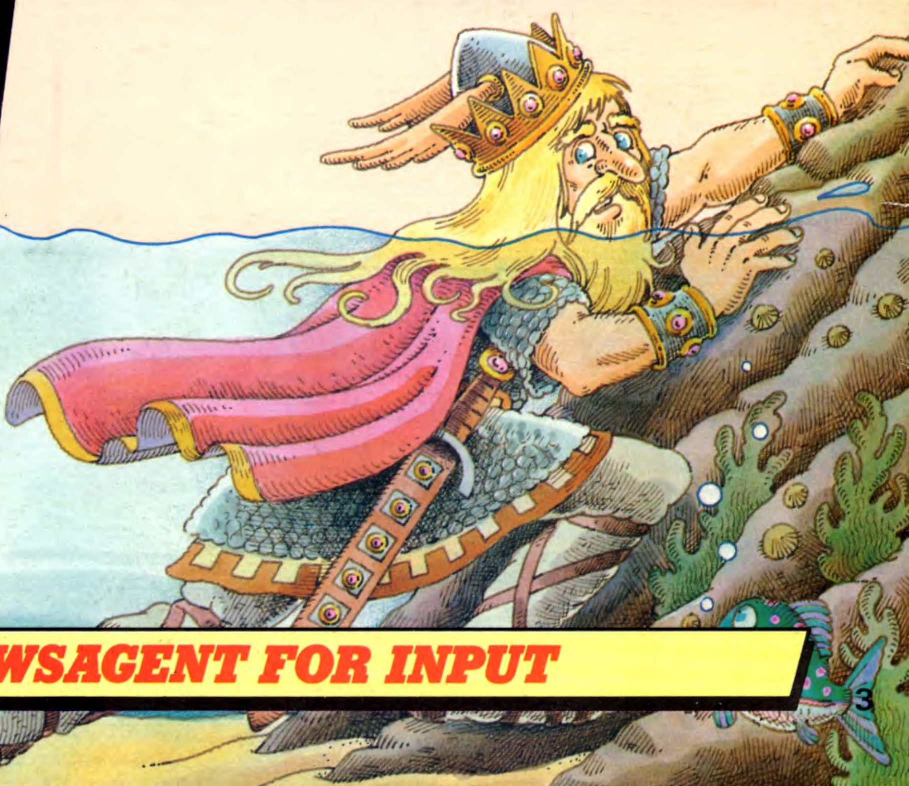
An interim index will be published each week. There will be a complete index in the last issue of *INPUT*.

COMING IN ISSUE 39...

- ❑ *Complete the BASIC arcade game by adding animation to FREDDIE AND THE SPIDER FROM MARS*
- ❑ *Put the squeeze on the DATA needed for your computer concerto with MUSIC COMPRESSION techniques*
- ❑ *Use new COMPUTER MODELS to try your hand at simulating a real-life business situation—running a food stall*
- ❑ *In CLIFFHANGER, add in the routines which control the RISING TIDE that is threatening our hero*
- ❑ *PLUS, for COMMODORE users, more routines for your ASSEMBLER*
- ❑ *Also in this issue, a comprehensive INDEX to parts 27–39 of INPUT*







ASK YOUR NEWSAGENT FOR INPUT